RASM: Roudoudou's Z80 Assembler V1.5

Édouard BERGÉ (Roudoudou)

July 19, 2021

Contents

1 Introduction

During the making of my first big demo, CRTC^3, available assemblers on CPC were too slow and too limited for my needs: The source code consisted in 250'000 words (out of comments), 35'000 labels and 60'000 expressions. I needed a damn fast assembler with cartridge management, memory bank management and integrated crunched code and data sections, so i had to write a new assembler.
It was not my first attempt to write such a tool: 18 years ago, i had already written a small assembler. It was so limited, i couldn't use it for CRTC^3, but i've kept some aspects of its design, particularly the fact that the process of assembling was mono-pass.
RASM uses efficient algorithmic patterns, such as merkel trees, caches, and grouped memory allocation. Thanks to its linear conception, performances in real conditions are really high, it's particularly fast, even with huge projects.

Nowadays RASM is used on big projects such as:

- Ghost'n Goblins by Golem13 - still in development, for CPC+ architectures.
- Arkos Tracker II by Targhan (embedded in the software)

1.1 Features

- Ultra-fast compilation
- common compressors included
- binary, snapshot, floppy, cartridge export
- symbols import/export
- overwriting control
- unlimited memory spaces where labels are shared
- dedicated directives for memory/ROM management
- all Z80 instructions are supported
- macros, conditionnal code, unlimited loops, local labels, switch/case
- double precision calculation with correct rounding
- audio file import, with automatic conversion into DMA lists (CPC+)
- optionnal compatibility with Maxam or AS80
- code duration evaluation

Rasm is licenced under MIT licence.

This documentation is maintained by Stephane Sikora, please send any feedback to
`sikoogle+rasm@gmail.com` .

2 Usage

RASM is meant to be convenient to use. It selects automatically the correct file format (plan binary, snaphost or cartridge file)) depending on the selected ROMs, give a consistent name to generated files, looks for files in relatives path, allows multiple origin (ORG) directives, but will detect overlapping code blocs, and so on...
RASM pre-processor not only performs some checks (valid characters, strings), it also converts some operators into their C equivalent (for example `XOR`, `AND`,`OR`,`MOD` which are used by Maxam), so you can indiferently use `AND` or `&` in your expressions. And last, but not least, it is possible to use conditional expressions in order to change the way your program is assembled (like with C preprocesser), and you can even define macros and data structures!

2.1 Command line

Usage:

```
RASM.exe <file to assemble> [options]
```

Without any option,

```
rasm.exe myfile.asm
```

produces `rasmoutput.bin` file.

2.2 Exported file names

These options are used for changing the name of the files produced by RASM:

- -o <file radix>: set a common name for each output file, disregarding its type (.bin, .sym, ...). The default value is "RASMoutput"
- -ob <binary filename>: set the full filename for automatic binary output.
- -os <symbol filename>: set the full filename for symbol output.
- -oa : generates output files (cpr,bin,sna) using the same name as the input source file
- -no : disable file output.

2.3 Symbol exports

Symbols such as label's addresses, constants, can be exported in a .sym file, using different formats. For example, it can be useful for debugging a program with Winape.

- -s : export symbols in RASM format
- -sw : export symbols in Winape format
- -sp : export symbols in Pasmo format
- -sl : also export local labels.
- -sv : also export variables.
- -sq : also export EQU aliases.
- -sx : export for ZX emulators, where the bank is printed : `(<bank>:<adresse>)`

- -sa : export all symbols (same as -sl -sv -sq)

Example:

```
RASM.exe test -o grouik -s
Pre-processing [test.asm]
Assembling
Write binary file grouik.bin (25 bytes)
Write symbol file grouik.sym (10 bytes)
```

The symbol file looks like this:

```
LABEL1 #0 B0
LABEL2 #1 B0
LABEL3 #2 B0
LABEL4 #4 B0
```

With -sp option:

```
LABEL1 EQU 00000H
LABEL2 EQU 00001H
LABEL3 EQU 00002H
LABEL4 EQU 00004H
```

With -sw, symbols are exported in Winape format:

```
LABEL1 #0
LABEL2 #1
LABEL3 #2
LABEL4 #4
```

It is possible to disable symbol export for portion of codes with `NOEXPORT` directive.

2.4 Including files

Including source code or binary file can be achieved with `INCBIN` and `READ` directives. By default, included files are searched in the local folder, paths are relative, but it is possible to specify one ore more folder where to look for files. It is also possible to define or import symbols:

- -I <include directory>: set include directory. Multiple options -I is possible. -l <filename >imports symbols in RASM, Pasmo or Winape format. You may import as many symbol file as you want using this option multiple times
- -D <var>=<value>: With this option you can define a variable. For example `-DFOO=1` will define `'FOO'` variable with value 1.
- -l <fichier label>: You can import labels from a file with this option. Various formats are supported (RASM, Sjasm, Pasmo or Winape formats) and are automatically detected. This option can be used many times.

```
RASM.exe test -l import1.sym -l import2.sym -l import3.sym
```

2.5 Dependencies options

- -depend=make Export all dependencies in a single line (for makefile usage)
- -depend=list Export all dependencies, one per line (for other usage)

If a filename for binary output is set (-ob option), it will be added to the dependances list, in first position.

2.6 Compatibility options

- texttt-m : Maxam compatibility:

  - unsigned 16 bit computation, with wrong rounding

  - comparisons are done with equal (=) sign

  - Operator priorities are simplified (see charts)

- texttt-amper : Used with maxam mode (`-m`). All # used for encoding hexa numbers will be replaced by & symbol.

- texttt-ass : AS80 compatibility:

  - 32 bits integer calculations with wrong rounding

  - DEFB,DEFW,DEFI or DEFR directives with more than one parameter use the address of the first byte, when using $

  - MACRO directive must be used after the name of the macro, and not before (see 5.3.3)

  - Macro parameters are not protected by {}

- texttt-uz : UZ80 compatibility:

  - 32 bits integer calculations with wrong rounding

  - macro parameters are not protected by

  - MACRO directive must be used after the name of the macro

- texttt-dams : DAMS compatibility:

  - labels beginning by a dot are ignored

- texttt-pasmo : PASMO compatibility:

  - DEFB,DEFW,DEFI or DEFR directives with more than one parameter use the address of the first byte, when using $

2.7 Debug options

These options are useful in rare occasions, it can sometime help for tracking bugs:

- -v : verbose mode, display stats
- -void : enforce usage of (void) syntax for macros without parameter (**??**)
- -wu : Display a warning when an alias, a variable or a label is declared, but not used
- -d : verbose detailed pre-processing
- -a : verbose detailed assembling
- -n : display third parties licences
- -xpr : for extended cartdidge export, generate additional files for every 512KB slot (**??**)

2.8 More options

Additional options are useful for some architectures only. Export optios specific to the Amstrad CPC (Snapshot and cartridge generation) are documented in paragraph 7.4.3. Also remember -help option, which will display a complete list of all available command line arguments.

3 Source code format

RASM is meant to be easy to use, it offers some flexibility, concerning the syntax. For example:

- It's useless to use indentation with Rasm, except for aesthetics purpose.
- There is no need for ':' suffix for identifying labels, but it is allowed, it will simply be ignored.
- Windows and Unix file format are both supported.
- RASM is not case sensitive, the whole source code is converted into upper case, so don't be surprised if you see your code in upper cases in error messages.

Simply keep in mind you cannot use a reserved word (directive, register, Z80 instruction) as a label. In this part,we'll see general syntax of a z80 assembly file, as it can be commonly found in other assemblers. In the next parts, we'll see some specific aspects of RASM.

3.1 Comments

Rasm uses semicolons to start a comment: Any character after a semicolon (until then end of the line) will be ignored. It also works possible to use C syntax, single line comments starting with a double slash (`//`), and multi lines comments delimited by `/*` and `*/`.

3.2 Labels

Labels are used for naming a specific memory adress.

```
ld HL,monlabel
call aFunction
aFunction:
; ...
ret
monlabel db 0
```

We'll see special labels later in this document: Some labels can be defined as locals to a Loop or macro (see local and proximity labels, 5.4.1). Also any label beginning with `BRK` or `@BRK` will generate a `BREAKPOINT`. (see section 7.4.2).

3.3 Z80 Instructions

The complete Z80 instruction set is supported, including undocumented ones. See Appendix B for the full opcode list.

3.3.1 IX, IY registers

IX and IY registers can be addressed as two 8 bit registers. For example, IX lower part can be addressed indifferently with LX, IXL or XL, and higher part with HX, IXH or XH:

```
ld A,IXL
```

Also, complex instructions with IX and IY are written with this syntax:

```
res 0,(IX+d),A
bit 0,(IX+d),A
sll 0,(IX+d),A
rl 0,(IX+d),A
rr 0,(IX+d),A
```

3.3.2 Undocumented opcodes syntax

```
out (<byte>),a
in a,(<byte>)
in 0,(c)
in f,(c)
sll <register>
sl1 <registre>
```

3.3.3 Shorcuts

Rasm allows some shortcuts. These are not real instrctions, but a convenient way to write shorter assembly code.

- Multi-arg PUSH and POP

  PUSH BC,DE,HL → PUSH BC : PUSH DE : PUSH HL
  POP HL,DE,BC → POP HL : POP DE : POP BC

- NOP repetition

  nop 4 → nop : nop : nop : nop

- Complex LD

  LD BC,BC → LD B,B : LD C,C
  LD BC,DE → LD B,D : LD C,E
  LD BC,HL → LD B,H : LD C,L
  LD DE,BC → LD D,B : LD E,C
  LD DE,DE → LD D,D : LD E,E
  LD DE,HL → LD D,H : LD E,L
  LD HL,BC → LD H,B : LD L,C
  LD HL,DE → LD H,D : LD L,E
  LD HL,HL → LD H,H : LD L,L

- Complex LD with IX,IY

  LD HL,(IX+n) → LD H,(IX+n+1) : LD L,(IX+n)
  LD HL,(IY+n) → LD H,(IY+n+1) : LD L,(IY+n)
  LD DE,(IX+n) → LD D,(IX+n+1) : LD E,(IX+n)
  LD DE,(IY+n) → LD D,(IY+n+1) : LD E,(IY+n)
  LD BC,(IX+n) → LD B,(IX+n+1) : LD C,(IX+n)
  LD BC,(IY+n) → LD B,(IY+n+1) : LD C,(IY+n)

  LD (IX+n),HL → LD (IX+n+1),H : LD (IX+n),L
  LD (IY+n),HL → LD (IY+n+1),H : LD (IY+n),L
  LD (IX+n),DE → LD (IX+n+1),D : LD (IX+n),E
  LD (IY+n),DE → LD (IY+n+1),D : LD (IY+n),E
  LD (IX+n),BC → LD (IX+n+1),B : LD (IX+n),C
  LD (IY+n),BC → LD (IY+n+1),B : LD (IY+n),C

- Alternative syntax

  EXA → EX AF,AF'

3.4 Memory related directives

3.4.1 ORG Directive

```
ORG <logical address>[,<physical address>]
```

ORG is used for locating assembled code to a specific address. This directive can be used multiple times in the same memory space, but assembled memory blocs may not overlap. In that case, RASM will produce an error message.

```
ORG #8000
RET
; bytecode output:
; #8000:  #C9
```

Still, if you need to generate two or more pieces of code targetted for the same address, but physically stored in a different place, you can use the second parameter. For example, in order to generate code targetted for address #8000, but stored in #1000:

```
ORG #8000,#1000
label:  JP label
; bytecode output:
; #1000:  #C3,#00,#80
```

On Amstrad CPC, you also can write tagetted to the same adress, by defining a new memory space, using BANK directive. (See section 7.3.5)

3.4.2 ALIGN

`ALIGN <boundary>[,fill]`

If the current memory address is not a multiple of the 'boundary' parameter, it will be increased in order to meet the alignment constraint. The gap between the current memory address and the aligned one is filled by zeroes, except if the second parameter is specified. For example:

```
ORG #8001
ALIGN 2 ; align code on even address (#8002)
ALIGN 256,#55 ; align code on high byte (#8100)
; #8002-#80FF is filled with #55
```

3.4.3 LIMIT

`LIMIT <address boundary>`

By default, the upper address for locating code is set to 65535, but for some reason, you may need to reduce thise value. Hoever if you want to protect a memory area zone, take a look at the `PROTECT` directive below.

3.4.4 PROTECT

`PROTECT <start address>,<end address>`

This directive protects a memory zone, delimited by the two parameters, from writing.

3.4.5 CONFINE

`CONFINE size [,warning]`

```
; OK - nothing is happening
ORG 250
CONFINE 6
tab1:  defs 6
```

```
; will align tab1 and produce a warning
ORG 250
CONFINE 10,warning
tab1:  defs 6
; Warning :  Confinement overflows 3 bytes
```

3.5 Data definition

3.5.1 DB, DEFB, DM, DEFM

```
DEFB <value1>[,<value2>,...]
DEFM <value1>[,<value2>,...]
```

This directive handle one or more parameters and output bytes regarding of thoses parameters. The value may be a literal value, a formula (the result will be rounded), or a string where each char will output a byte. Following code will producte 'Roudoudou' string. ('u' char corresponds to ASCII code #75) Example:

```
org #7500
label:
defb 'r'-'a'+'A','oudoud',#6F,hi(label)
```

With character strings, it's possible to use control characters, with
, just like in C syntax:
n
t
r
See CHARSET directive for altering the way strings are interpreted.

3.5.2 DEFW

```
DEFW, DW <value1>[,<value2>,...]
```

This directive handles one or more parameters and output words (two bytes). Values may be literals, formula, single char, but char strings are not allowed!
Example:

```
DEFW mylabel1,mylabel2,'a'+#100
```

3.5.3 DEFI

```
DEFI <value1>[,<value2>,...]
```

This directive handles one or more parameters and output four bytes integers. Values may be literal value, formula, single char, but not a string!

3.5.4 DEFR

```
DEFR <real number1>[,<real number2>,...]
```

DEFR (or DR) directive handles one or more parameters and output AMSTRAD firmware compatible real numbers (5 bytes)
Example:

```
defr 5/12, 0.5, sin(90)
```

3.5.5 DEFS

`DEFS, DS <repetition>[,<value>,[<repetition>,...]`

This directive is used for repeating the same byte many times. If no output value is set, then zeroes will be written. If repetition value is zero then nothing will be output. You can declare more than one repetition sequences with only DEFS.
Examples:

```
defs 5,8,4,1   ; #08,#08,#08,#08,#08,#01,#01,#01,#01
defs 5,8,4     ; #08,#08,#08,#08,#08,#00,#00,#00,#00
defs 5         ; #00,#00,#00,#00,#00
```

3.5.6 STR

`STR 'string1'[,'string2'...]`

Almost same directive as `DEFB`, except the very last char will have its 7th bit set to 1. Both lines will output the same byte sequence:

```
str 'roudoudou'
defb 'roudoudo','u'+128
```

3.5.7 CHARSET

```
CHARSET
CHARSET 'string',<value>
CHARSET <code>,<value>
CHARSET <start>,<end>,<value>
```

This directive allows to redefine quoted char values to be changed. There are 4 ways to use this directive:

- 'string',<value>: First char of the string will be transposed as <value>. The next char as <value>+1 and so on, until the end of the string.
- <code>,<value>: replaces char with ASCII code <code>by a char with ASCII value <value>.
- <start>,<end>,<value>: replaces the characters with ASCII values within the range [<start>;<end>] by <value>,<value+1>, and so on.
- without parameter : Ignore any previous `CHARSET` directive: strings will stay unchanged.

For example, you can set a simple char redefinition:

```
CHARSET 'T','t' ; 'T' chars will be translated as 't'
DEFB 'tT' ; #74 #74
```

Or redefine consecutives chars in a range:

```
CHARSET 'A','Z','a' ; Change all uppercases to their respective lowercases
DEFB 'abcdeABCDE' ; #61,#62,#63,#64,#65,#61,#62,#63,#64,#65
```

You can also redefine non consecutives chars:

```
CHARSET 'turndiskheo ',0
DEFB 'there is no turndisk'
;#00,#08,#09,#02,#0B,#05,#06,#0B,#03,#0A,#0B,#00,#01,#02,#03,#04,#05,#06,#07
```

3.5.8 $ Operator

The symbols ($) refers to the current byte address. For example:

```
org #8000
defw $,$
```

is equivalent to:

```
defw #8000,#8002
```

With AS80 compatibility mode it would produce the same output as:

```
defw #8000,#8000
```

However, when used with ORG directive, $ refers to the physical address, not the logical one:

```
ORG #8000,#1000
defw $ ; #8000 is written in #1000
ORG $ ; ORG considers the physical address (#1002)
defw $ ; #1002 is written in #1002
; bytecode output:
; #1000:  #00,#80,#02,#10
```

4 Expressions

4.1 Aliases and Variables

EQU is a common way in assemblers to define constants in a convenient way. RASM also introduces variables, which value can be changed during the assembling process. There is no limit in the number of variables or alias that can be defined.

4.1.1 Constants or Alias

`<alias> EQU <replacement string>`

`EQU` directive allows to define aliases by associating a symbol with a value: any occurence of the symbol will be replaced by the value. An alias cannot be changed once it has been changed, it's a constant value. There is an infinite recursivity check done for each alias declaration.
Example:

```
tab1 EQU #8000
tab2 EQU tab1+#100
ld HL,tab2
```

4.1.2 Variables

An alias it cannot be changed, once it has been defined. However, it is possible to use variables with RASM, with the following syntax:

```
myvar=5
LET myvar=5 ; Winape compatible Variables are used for numeric values only.
```

You can define as many variables as you want. Some examples:

```
dep=0
repeat 16
ld (IX+dep),A
dep=dep+8
rend
```

```
ang=0
repeat 256
defb 127*sin(ang)
ang=ang+360/256
rend
```

4.2 Literal values

Rasm accepts these values in expressions:

- Decimal if the value begins with a digit.
- Binary if the value begins with
- Octal if the value begins with @.
- Hexadecimal if the value begins with #, $, 0x or ends with h.
- ASCII value of a char, delimited by quotes.

- Value of a constant or a variable, referenced by its name, eventually prefixed with '@'.
- Current address symbol ($)

All internal calculation are done with double precision floating point accumulator. A correct rounding is done in the end for integer needs. If the evaluation leads to a computation error, the result will be null.
Beware of the & char, it is reserved for AND operator.

4.2.1 Allowed chars

Between quotes, all standard ASCII characters are allowed. Quoted strings may contains escaped chars: `\t \n \r \f \v \b \0` . Escaped characters are ignored when used with PRINT directive.

4.3 Operators

Rasm is using a simplified calculation engine with multiple priorities (like C language). Here is the list of supported operations:

*	multiply	/	divide
+	addition	−	subtraction
^ or XOR	logical Exclusive OR	%% or MOD	Modulo
& AND	Logical AND	| OR	Logical OR
&&	Boolean AND	||	Boolean OR
<<	Left shift (multiply by 2^n)	>>	Right shift (divide by 2^n)
`hi()`	get upper 8 bits of a word	`lo()`	get lower 8 bits of a words
`sin()`	sinus	`cos()`	cosinus
`asin()`	arg sinus	`acos()`	arc-cosinus
`atan()`	arc-tangente		
`int()`	float to integer conversion	`frac()`	keeps fractional part of a float
`floor()`	rounds to the lower integer	`ceil()`	ronds to the higher integer
`abs()`	absolute value	`rnd()`	Random number between 0 and $n-1$
`ln()`	neperian logarithm	`log10()`	base 10 logarithm
`exp()`	exponent	`sqrt()`	square root
==	equals (= in Maxam mode)	! = ou <>	not equal
<=	lesser or equal	>=	greater or equal
<	lesser	>	greater

4.4 Operators priorities

Lower is the prevalence, higher is the execution priority.

Operators	Rasm Prevalence	Maxam Prevalence
( )	0	0
!	1	464
* / %	2	464
+ −	3	464
<< >>	4	464
< <= == => > ! =	5	664
& AND	6	464
| OR	7	464
^ XOR	8	464
&&	9	6128
||	10	6128

5 Preprocessor

RASM preprocessor recognizes many directives.
When a directive has parameters, it must be separated by at least one space char:
Wrong syntax: `ASSERT(4*myvar)`
Correct syntax: `ASSERT (4*myvar)`

5.1 Debugging and asserting

5.1.1 PRINT

`PRINT 'string',<variable>,<expression>`

Write text, variables or the result of evaluation of an expression during assembly.
By default, numerical values are formatted as floating point values, but you may use prefixes to change this behaviour:

- {hex} Display in hexadecimal format. If the value is less than `#FF` two digits will be displayed. If less than `#FFFF`, the display will be forced to 4 digits.
- {hex2}, {hex4}, {hex8} to force hex display with 2, 4 or 8 digits.
- {bin} Display a binary value. If the value is less than `#FF` 8 bits will be displayed. Otherwise if it is less than `#FFFF` 16 bits will be printed. Any negative 32 bits value with all 16 upper bits set to 1 will be displayed as a 16 bits value.
- {bin8},{bin16},{bin32} Force binary display with 8, 16 or 32 bits.
- {int} Display value as integer.

5.1.2 FAIL

`FAIL 'string',<variable>,<expression>`

This directive is similar to `PRINT`, but it will also trigger an error and STOP assembling.

5.1.3 STOP

Stop assembling an do not generate any file.

5.1.4 NOEXPORT

```
NOEXPORT [Symbols]
ENOEXPORT [Symbols]
```

`NOEXPORT` directive disables symbol export. By default, it applies to all symbols (labels,variables,constants), but it is possible to specify a subset of symbols. Symbol export can be re enabled (fully or partially) with `ENOEXPORT`.

5.2 Conditionnal code directives

It is possible to use conditional directives with RASM, in a way similar to C preprocessor: it is possible to change the assembled code, depending on some conditions. There is a basic rule when writing such expressions: all variables used in it must be declared prior to the expression.

5.2.1 ASSERT

```
ASSERT <condition>[,text,text,text...]
```

Stop assembling if the condition test fails. In that case, and if some text is specified, it will be printed on the console. Example:

```
assert mygenend-mygenstart<#100
assert mygenend-mygenstart<#100, 'code is too big'
```

5.2.2 IF, IFNOT

```
IF <condition> ... [ELSE ...] ENDIF
IF <condition> ... [ELSEIF <condition> ...] ENDIF
IFNOT <condition> ... [ELSE, ... ] ENDIF
IFNOT <condition> ... [ELSEIF <condition> ...] ENDIF
```

As with C preprocesor, this directive can be used for enabling some portions of code, depending on a condition. Example:

```
CODE_PRODUCTION=1
[...]
if CODE_PRODUCTION
 or #80
else
 print 'test version'
endif
```

5.2.3 IFDEF, IFNDEF

```
IFDEF <variable or label> ... [ELSE ... ] ENDIF
IFNDEF <variable or label> ... [ELSE ... ] ENDIF
```

Both directives test variable or label existence.

5.2.4 UNDEF

```
UNDEF <variable>
```

Removes a variable definition. Any `IFDEF` condition with this variable will be evaluated as false. If the variable doesn't exist, this directive won't do anything.

5.2.5 IFUSED, IFNUSED

```
IFUSED <variable or label> ... ENDIF
IFNUSED <variable or label> ... ENDIF
```

Both directives test variable or label usage, BEFORE the test.

5.2.6 SWITCH

`SWITCH/CASE` syntax mimics the C syntax. A `SWITCH` block is terminated by `ENDSWITCH` directive,and each of its `'CASE'` block with a `'BREAK'`. With RASM, you can use the same value in different cases, allowing to write more complex cases. For example, this code will be produce 'BCE' string:

```
myvar EQU 5

switch myvar
  nop ; outside any case, will never be evaluated
 case 3
  defb 'A'
 case 5
  defb 'B'
 case 7
  defb 'C'
  break
 case 8
  defb 'D'
 case 5
  defb 'E'
  break
 default
  defb 'F'
endswitch
```

5.3 Loops and Macros

5.3.1 REPEAT

`REPEAT <number of repetitions>[,counter[,counter_start[,counter_step]]] ... REND`
`REPEAT ... UNTIL <condition>`

This directive repeats a block of instructions. You may fix a number of repetition or use conditionnal mode with `UNTIL`. It is also possible to close such a bloc with `ENDREP` or `ENDREPEAT`, for Vasm compatibility.

```
cnt=90
repeat
 defb 64*sin(cnt)
 cnt=cnt-4
until cnt<0
```

Non Conditional Repeat In the case of a non conditional loop, it is possible to specify a variable which contains the iteraction counter. There's no need to declare this variable (here `cnt`) prior to the `REPEAT` block. It will automatically be created.

```
repeat 10,cnt
 ldi
 print cnt
rend
```

By default, the loop counter starts from 1 and is increased by 1 at every loop. These values can be changed, respectively with `counter_start` and `counter_step` optional parameters. These values can be integer, but also floating values.

```
repeat 26,letter,65
 db letter
rend
```

These values can either be integer, but also floating values.

```
repeat 180,angle,0,3.14/180
 db sin(angle)*64+64
rend
```

SymbolREPEAT_COUNTER

`REPEAT\_COUNTER [counter_start[,counter_step]]`

Another way to specify textttcounter_start and `counter_step` consists in using `REPEAT_COUNTER`. All REPEAT blocks declared after `REPEAT_COUNTER` will use these values. Without parameters, the counter will start at 1, and increment will be set to 1.

```
y=10
repeat 3,x,10
assert x==y
y+=5
rend
```

You can get the internal loop counter anytime with internal variable `REPEAT_COUNTER`.

```
repeat 10
 ldi
 print repeat_counter
rend
```

5.3.2 WHILE, WEND

`WHILE <condtion> ... wEND`

Repeat a block as long as the condition is evaluated as true. You may use the internal variable `WHILE_COUNTER` variable to get the loop counter.

```
cpt=10
while cpt>0
 ldi
 cpt=cpt-1
 print 'cpt=',cpt,' while_counter=',while_counter
wend
```

This code will loop 10 times with the following output:

```
Pre-processing [while.asm]
Assembling
cpt= 9.00 while_counter= 1.00
cpt= 8.00 while_counter= 2.00
cpt= 7.00 while_counter= 3.00
cpt= 6.00 while_counter= 4.00
cpt= 5.00 while_counter= 5.00
cpt= 4.00 while_counter= 6.00
cpt= 3.00 while_counter= 7.00
cpt= 2.00 while_counter= 8.00
cpt= 1.00 while_counter= 9.00
cpt= 0.00 while_counter= 10.00
Write binary file rasmoutput.bin (20 bytes)
```

5.3.3 Macros

```
MACRO <macro_name> [param1[,param2[,...]]]
...
MEND
```

A macro is a way to extend the language, by defining a block of instructions, delimited by MACRO and `MEND` (or ENDM) directives, that can later be inserted in your code. Macros can take parameters, so you can make conditionnal assembling with it: a macro is barely a copy/paste with arguments replacement. Arguments inside the macro are referenced using curly brackets. Here is an example of a long distance indexing, working for any 16 bit register (except B or C):

```
macro LDIXREG register,dep
 if {dep}<-128 || {dep}>127
  push BC
  ld BC,{dep}
  add IX,BC
  ld (IX+0),{register}
  pop BC
 else
  ld (IX+{dep}),{register}
 endif
mend

LDIXREG H,200
LDIXREG L,32
```

Macro invocation Beware that RASM will understand any misspeled macro as a label declaration! A recommanded usage for macros without parameters, is to systematicaly use an empty parameter "`(void)`" . A misspeled macro call will then trigger an error. If you want to enforce usage of this syntax, you can use `-void` option. Using a macro without any parameter will raise an error.

```
MACRO withoutparam
nop
MEND
withoutparam (void) ; secured call of macro
```

Macro calls with static or dynamic args Arguments sent to a macro can also be formulas.

```
MACRO test myarg
 DEFB {myarg}
MEND
;Same as  defb 1 :  defb 2:
REPEAT 2
 test repeat_counter
REND
;Same as  defb 1 :  defb 1:
REPEAT 2
 test {eval}repeat_counter
REND
```

Separating Low an Hi bytes of a 16bit register Inside a Macro, it is possible to use LOW or HI for using the lower or the higher part of a 16 bit register. For example, `ld A,R1.low` is identical to `ld A,C` if R1=BC. A macro for adding two 16 bits register can be written like this:

```
macro add16, R1, R2
 ld A,{R1}.low
 add {R2}.low
 ld {R1}.low,A
 ld A,{R1}.high
 adc {R2}.high
 ld {R1}.high,A
mend
add16 bc,hl
```

5.4 Labels and modules

5.4.1 Local labels

Inside a loop (`REPEAT/WHILE/UNTIL`) or inside a macro, you can define local labels the same way as Winape assembler does, by prefixing labels with '@'. You cannot use these labels outside of the loop or macro.
You can use label value with a directive (`ORG` for example) only if the label was previously declared.
Usage of local label in a loop:

```
repeat 16
add hl,bc
jr nc,@no_overflow
dec hl
@no_overflow
rend
```

5.4.2 Proximity labels

Proximity labels are prefixed with a dot, and are associated with the previous label. They can be used 'locally' directly with their 'short' names, and anymwhere with their full name:

```
routine1:
 add hl,bc
 jr nc,.no_overflow
 dec hl
.no_overflow
routine2:
 add hl,bc
 jr nc,.no_overflow
 dec hl
.no_overflow
routine3:
 xor a
 ld hl,routine1.no_overflow ; retrieve proximity label of routine1
 ld de,routine2.no_overflow ; of routine2
 sbc hl,de
```

5.4.3 Mixing different kinds of labels

```
global:  nop
.prox:  nop ; (1)

repeat 2
 jp .prox ; (=> 1)
@label:  nop ; (2)
.prox :  nop ; (3)
@label2:  nop ; (4)
.prox :  nop ; (5)
 jp global.prox ; (=> 1)
 jp @label ; (=> 2)
 jp @label.prox ; (=> 3)
 jp @label2.prox ; (=> 5)
 jp .prox ; (=> 5)
rend

 jp .prox ; (=> 1)
 jp global2.prox ; (=> 7)

global2:  nop ; (6)
 jp .prox ; (=> 7)
.prox:  nop ; (7)
 jp global.prox ; (=> 1)
 jp global2.prox ; (=> 7)
 jp .prox ; (=> 7)
```

5.4.4 Modules

```
MODULE <namespace>
...
[MODULE OFF]
```

`MODULE` is a way to declare a section in your code. It is similar to namespace in C, and allows to prefix globals and proximity labels with a name. It can be usefull in order to avoid conflict between portions of code using similar labels, for example when including external code. Closing a Module section can be done by declaring a new module, or using `MODULE OFF` In order to prefix a label inside a module, underscore symbol is used : `module_label`. For exemple:

```
MODULE module1:
start:
...
ret
data:  equ 1

MODULE module2:
start:
...
ret
data:  equ 2
MODULE OFF

ld a,(module1_data)
call module2_start
```

5.5 Structures

5.5.1 STRUCT

```
STRUCT <prototype name> [,<variable name>]
...
ENDSTRUCT
```

As Z80 processor is able to manage structured data thanks to its IX and IY registers, RASM introduces STRUCT directive, wich is used for defining a structure, in a similar way to C syntax

```
; structure st1 created with two fields ch1 and ch2.
struct st1
ch1 defw 0
ch2 defb 0
endstruct
; Nested structures:
; metast1 is created with 2 sub-structures st1 called pr1 et pr2
struct metast1
struct st1 pr1
struct st1 pr2
endstruct
```

When {STRUCT} directive is used with 2 parameters, RASM will create a structure in memory, based on the prototype. In the example below, it will instantiate a metast1 structure type, called mymeta.

```
struct metast1 mymeta
```

Example of retrieving fields absolute address using the structure previously declared:

```
LD HL,mymeta.pr2.ch1
LD A,(HL)
```

Example of accessing a field with an offset, by using the prototype name:

```
LD A,(IX+metast1.pr2.ch1)
```

5.5.2 SIZEOF

Recommended usage to get the size of a structure is to use {SIZEOF} prefix. It alsoworks for a substructure or a field.

```
LD A,{SIZEOF}metast1 ; LD A, 6
LD A,{SIZEOF}metast1.pr2 ; LD A, 3
LD A,{SIZEOF}metast1.pr2.ch1 ; LD A, 2
```

Like Vasm, you also can get the structure size using its prototype name but it is not recommended.

5.5.3 STRUCT Array

It's possible to instanciate an array of structs, using this syntax:

```
struct mystruct my_instances,10
```

Compared to `ds 10*SIZEOF(mystruct)`, data is initialized with default values as defined in structure declaration, and not filled with a zero value. Also it is possible to access to a specific instance, using an index, like a regular array.

```
| LD HL,myinstances5
```

5.6 Duration of a bloc

```
TICKER START,<var>
...
TICKER STOP|STOPZX,<var>
```

`TICKER` directive computes the duration of an instruction bloc (delimited by TICKER START and TICKER STOP), and stores the result in a variable. It can be used for counting cycles for CPC architecture (by using `TICKER STOP`), or for ZX architectures (by using `TICKER STOPZX`) On CPC, the duration is expressed as "number of NOPs", which is approximatively equivalent to micro seconds (see Annexe C). This directive is very convenient when writing video effects, such as rasters, where colors have to be changed periodically, every 64 micro seconds (the duration of a video line):

```
| ld hl,col_tab
| ld bc,col_port ;#7fxx
| out (c),c
| ld d,20
| loop:
| TICKER START, cntline
|  ld a,(hl)
|  out (c),a
|  inc hl
| TICKER STOP, cntline
|  ds 64-4-cntline
|  dec d
|  jr nz, loop
```

6 Crunch and import directives

6.1 File Import

6.1.1 INCLUDE

```
INCLUDE 'file to read'
READ 'file to read'
```

Read a textfile in place of the directive. The root of the relative path is the location of the file containing the include directive. An absolute path discard the relative path. There is no recursivity limit, so be aware of what you are doing.

6.1.2 INCBIN

```
INCBIN 'file to read'[,offset[,size[,extended offset[,OFF]]]]
INCBIN 'file to read',REVERT
INCBIN 'file to read',REMAP,numcol
INCBIN 'file to read',VTILES,numtiles
INCBIN 'file to read',ITILES,width
```

Read a binary file. Binary data will go straight to memory space. Additional parameters are an offset, a size, and an option for disabling overwrite check. The extended offset is only here for compaitibily with Winape, so you can ignore it.

- You may use a negative size, for omitting some bytes at the end of the file: With a size of -10, the whole file except the 10 last bytes will be included.
- A null size will read the whole file.
- You may use a negative offset, it will be relative to the end of the file.
- The 'OFF' parameter will disable overwrite check for this file. You may want to read binary data in order to initialise a memory space, then assemble code on it.

Example:

```
ORG #4000
INCBIN 'makeraw.bin',0,0,0,OFF ; read in #4000, overwrite check is disabled

ORG #4001
DEFB #BB ; overwrite 2nd byte (in #4001) without error
```

If you want for example to import a 32K file into 2 16Kb banks (see 7.3.5)

```
bank n
incbin 'my32Kbfile.bin',0,16384
bank n+1
incbin 'my32Kbfile.bin',16384,16384
```

6.1.3 Multiple files import

You can use `INCBIN` inside a `REPEAT` block, for importing a series of files. For example if you want to import files myfile1, myfile2, ... myfile10:

```
REPEAT 10,cpt
INCBIN 'myfile{cpt}'
REND
```

If REVERT keyword is used, the file will be inserted backwards.
REMAP VTILES and ITILES variants are used for importing sprites.

6.1.4 Audio Files

```
INCBIN Filename,SMP|SM2|SM4
INCBIN Filename,DMA,preamp,[Option1[,Option2[,...]]]
```

All WAV formats -single channel or multi channel- are supported . Voices will be merged in latter case. The sampling frequency is not taken into account. To import a WAV file, you must specify one fo the 4 formats among SMP, SM2, SM4 and DMA:

- With SMP format, a sample corresponds to a byte.
- SM2 format groups two values in a single byte (two nibbles), the first sample corresponding to the 4 most significant bits.
- SM4 format four samples are stored in a single byte, the first sample being stored in the 2 most significant bits. 2 bits values are converted to 4 bits as follow: 00b → 0 ,01b → 13, 10b → 14, 11b → 15
- DMA format, which prepares data as DMA list, so it is specific to the CPC+ architecture family. DMA lists are ready to be executed by the PSG. Your audio file for DMA list must first be converted to 15600Hz. It is possible to specify a preamp factor, and also additional options:
  - `DMA_INT` : for triggering an interruption once the sample has been played
  - `DMA_CHANNEL_A` , `DMA_CHANNEL_B`, `DMA_CHANNEL_C` : For choosing which PSG channel to use
  - `DMA_REPEAT,count` : For repeating the sample up to 4095 times.

```
ORG #4000
INCBIN 'sound.wav',SMP
INCBIN 'sound.wav',DMA,1,DMA_CHANNEL_A,DMA_INT,DMA_REPEAT,4
```

6.2 Crunching

6.2.1 Crunched Section

```
LZ48 | LZ49 | LZ4 | LZX7 | LZEXO | LZAPU |LZSA1 [minmatchsize] | LZSA2 [minmatchsize] | LZX0 | LZX0B
...
LZCLOSE
```

Open a crunched section in LZ48,LZ49, LZ4, ZX7, LZAPU, LZSA, LZX0, or Exomizer. A LZ section is closed with `LZCLOSE`.
LZSA takes an optional parameter, for controlling how strong data will be crunched, and how fast it will uncompress. For example, for LZSA1, with minmatch=5, it will uncrunch quickly For LZSA2, with minmatch=2, it will crunch strongly. For more information, check the documentation by Emmanuel Marty, who created LZSA cruncher.
LZX0 is a new recent addition to RASM, and offers a good compression ratio, and uncrunches realy quickly. LZX0b is a variant for decrunching backwards.
Generated code is crunched once it is assembled. The code following such a block is then relocated (labels, ...).

You cannot call a label located after a crunched zone from the crunched zone because RASM cannot determine where it will be located after crunching. This will trigger an error.
Code or data of a crunched zone cannot exceed 64K. Also, you cannot imbricate crunched sections.
Example:

```
org #1000
ld hl,crunchedsection
ld de,#8000
call decrunch
call #8000
jp next ; label next after crunched zone will be relocated

crunchedsection:
LZ48 ; -- this section will be crunched
org #8000,$
nop
nop
nop
ret
LZCLOSE ; -- end of crunched section

next:
ret
```

6.2.2 Crunched Binaries

`INCL48, INCL49, INCLZ4, INCZX7, INCEXO, INCAPU, INCLZSA1, INCLZSA2, INCZXO, INCZXOB 'file to read'`

Read a binary file, crunch it in LZ48, LZ49, LZ4, Exomizer, LZSA or ZX7 on the fly.

6.2.3 SUMMEM

`SUMMEM start_address,end_address`

This directive sums all bytes beween `start_address` and `end_address` in the current bank and store the result at the address where the directive is located.

6.2.4 XORMEM

`XORMEM start_address,end_address`

Same as `XORMEM`, but computes a XOR operation instead of a sum. It can be used for computing a checksum, for example:

```
checkrom:
xor a
ld hl,0
ld bc,#1000
.computexor:
xor (hl)
inc hl
ld d,a
dec bc
ld a,b
or c
```

```
ld a,d
jr nz,.computexor
ld hl,checksum
cp (hl)
jr nz,romKO
jr romOK
checksum:
xormem 0,#1000
```

7 Amstrad CPC Specific features

7.1 Bank Management

7.1.1 BANK Prefix

Using {BANK} prefix before a label (example: {BANK}mylabel) will return the BANK number where the label is located, intead of its absolute address. For example:

```
BANK 0
ld a,{bank}mysub ; will be assembled as LD A,1
call connect_bank
jp mysub

BANK 1
defb 'hello'
mysub
jr $
```

7.1.2 PAGE Prefix

Use PAGE prefix before a label (example: {PAGE}mylabel) to get a value that can be used to program the Gate Array for accessing the bank where the label is located. For example with a label located into BANK 5, #7FC5 will be returned. If you are using BANKSET directive to select 4 banks in a 64K set, then the gate array value is composed by the set number and the 2 most significant bits of the label address. Example:

```
BANK 0
 ld bc,{PAGE}mysub ; will be assembled LD BC,#7FC5
 out (c),a
 jp mysub

BANK 5
 defb 'hello'
mysub
 jr $
```

7.1.3 PAGESET Prefix

You can use {PAGESET} prefix before a label (example: {PAGESET}mylabel) to program the Gate array for selecting the BANKSET where the label is located. For example, for a label stored in BANK #5, #7FC2 will be returned.

```
BANK 0
 ld a,lo({pageset}mysub) ; will be assembled as LD A,#C2
 ld b,#7F
 out (c),a ; whole RAM is switched, code is supposed
 jp mysub ; to be stored in ROM, or in a proper place

BANK 5
 defb 'hello'
mysub
 jr $
```

7.2 AMSDOS headers and DSK files

7.2.1 AMSDOS Header

This directive adds an AMSDOS header to the binary file generated by RASM. This directive has no effect on `SAVE` directive, which has its own option for adding AMSDOS header.

7.2.2 SAVE directive

`SAVE 'filename',<address>,<size>[,AMSDOS|DSK|TAPE['filename'[,<side>]]]`

Records a binary file of the given size, starting from the specified address, from current memory space. All `SAVE` directives are executed at the end of the assemblig process: there is no way to save intermediate assembling states.
When recording a file on a floppy image (DSK), its name will be automatically converted according to the AMSDOS format: lower cases will be replaced by upper cases, and it will be truncated. If the DSK file doesn't exist, il will be automatically created. If it already exists, and if the binary file produced by rasm already exists on the disk, it WON'T be updated, except if `-eo` option is used.
With TAPE format, a CDT filt will be produced.
Examples:

```
;Save a raw binary file
SAVE 'myfile.bin',start,size

;Save a binary file with AMSDOS header
SAVE 'myfile.bin',start,size,AMSDOS

;Save a binary file (AMDOS header mandatory) on a DSK file
SAVE 'myfile.bin',start,size,DSK,'fichierdsk.dsk'
```

Combined with RUN:

```
ORG #9000

start:
 call #bb06
 ret
end:

RUN start
SAVE 'main.bin',start,end-start,DSK,'main.dsk'
```

7.3 Snapshot and Cartridges

RASM also allows to generate cartridge (.crt) and snapshot (.sna) files. These files can be used by some emulators such as Wanape and Ace.

```
BUILDCPR [EXTENDED]
BANK 0
```

Cartridge Generation Without parameter, this directive is optional, as by default, when a BANK directive is used, a cardridge file is generated. However, it is recommended to explicitly use this directive to indicate that a cardridge will be generated. If `EXTENDED` parameter is added, then an extended cartdridge (.xpr) will be generated. It can be used in cunjunction with `-xpr` option, for generating additional file for each 512KB slot.

```
BUILDSNA
BANK 0
RUN #A000
```

Snapshot Generation

7.3.1 BUILDSNA

`BUILDSNA [V2]`

This directive forces Rasm to generate a snapshot instead of a cartridge. The entry point must be specified (0 is not valid).
By default, the snapshot is targeted for a CPC 6128 with CRTC 0. You can use `SETCRTC` and `SETCPC` directives to select an other configuration. Audio channels are disabled, ROMS are disabled and interrupt mode is set to 1.

7.3.2 SETCPC

`SETCPC <model>`

Select CPC model when recording a v3 snapshot:
0 : CPC 464
1 : CPC 664
2 : CPC 6128
4 : 464 Plus
5 : 6128 Plus
6 : GX-4000

7.3.3 SETCRTC

`SETCRTC <CRTC model>`

Select CRTC model when writing a v3 snapshot file. Value for CRTC model ranges from 0 to 4. CRTC 3 corresponds to CPC Plus and GX-4000, othe values to classic CPCs.

7.3.4 SETSNA

```
SETSNA RegisterName,value
SETSNA GA_PAL,index,value
SETSNA CRTC_REG,index,value
SETSNA PSG_REG,index,value
```

With first syntax (taking two parameters), these registers can be set:

- Z80 Main registers: Z80_AF, Z80_F, Z80_A, Z80_BC, Z80_C, Z80_B, Z80_DE, Z80_E, Z80_D, Z80_HL, Z80_L, Z80_H,
- Z80 Mirror registers: Z80_AFX, Z80_FX, Z80_AX, Z80_BCX, Z80_CX, Z80_BX, Z80_DEX, Z80_EX, Z80_DX, Z80_HLX, Z80_LX, Z80_HX
- Z80 Internal registers: Z80_R, Z80_I, Z80_IFF0, Z80_IFF1, Z80_IX, Z80_IXL, Z80_IXH, Z80_IY, Z80_IYL, Z80_IYH, Z80_SP, Z80_PC, Z80_IM,
- Gate Array: GA_PEN, GA_ROMCFG, GA_RAMCFG, GA_VSC, GA_ISC
- CRTC internal registers: CRTC_SEL, CRTC_TYPE, CRTC_HCC, CRTC_CLC, CRTC_RLC, CRTC_VAC, CRTC_VSWC, CRTC_HSWC, CRTC_STATE,

- PPI: PPI_A, PPI_B, PPI_C, PPI_CTL, PSG_SEL, CPC_TYPE, INT_NUM,
- FDD: FDD_MOTOR, FDD_TRACK
- PRINT: PRNT_DATA
- INTERRUPTS : INT_REQ

With the 3 other syntaxes, `SETSNA` takes an additional parameter, in order to specify the index of the register

7.3.5 BANK

```
BANK [ROM page number]
BANK [RAM page number]
BANK NEXT
```

Selects a ROM bank (while exporting a cartridge) or a RAM slot (for snapshots) for storing code or data. For a cartdridge, values range from 0 to 31. In snapshot mode the values range from 0 to 35 (64K base memory + 512K extended memory). Used without parameter, BANK directive opens a new memory workspace.
By default, when using BANK, a cartridge will be generated, except if `BUILDSNA` directive was used previously.

```
BUILDSNA ; recommanded usage when using snapshot is to set it first
BANKSET 0 ; assembling in first 64K
ORG #1000
RUN #1000 ; entry point is set to #1000

 ld b,#7F
 ld a,{page}mydata ; get gate array value for paging memory
 out (c),a
 ld a,(mydata)
 jr $

BANK 6 ; choose 3th bank of 2nd 64K set
nop
mydata defb #DD

bank
; bank used without parameter, this is a temporary memory space
; that won't be saved in the snapshot

pouet
 repeat 10
  cpi
 rend
camion
 SAVE"another",pouet,camion-pouet
```

By default, snapshot v3 are exported. There is a compatibility option for selecting snapshot version 2, some emulators or hardware board do not support snapshot v3 yet. Just add arg 'v2' to `SNAPSHOT` directive or add -v2 option to the command line while invocating RASM.

7.3.6 RUN

```
RUN <address>[,<gate array configuration>]
```

This option is only used to set then entry point of a snapshot file. It is ignored if a cartridge is exported. The gate array can be configured with additional parameters

7.4 Specific Directives for snapshot images

7.4.1 BANKSET

```
BANKSET <64K bloc number>
```

BANKSET directive select a set of 4 pages in a row. With snapshot v3, there are 9 memory sets, indexed from 0 to 8.
You may use BANK and BANKSET in a source but you cannot select the same memory space. A check will trigger an error if you try to.
Using this directive enables snapshot output (like BUILDSNA does).

7.4.2 BREAKPOINT

```
BREAKPOINT [<address>]
[@]BRKlabel
```

Add a breakpoint (this won't be assembled) to the current address or to the address of the parameter. Breakpoints may be exported to a text file or into snapshots (Winape and ACE compatible) with -sb option.
Another way to set a breakpoint is to prefix a label with `BRK` or `@BRK`.

7.4.3 Export option

- -oc <cartridge filename>: set the full filename for cartridge output.
- -oi <snapshot filename >set the full name of exported snapshot file
- -v2 : Export a snapshot version 2 (default is version 3)
- -ss : Export symbols in snapshot file (Winape and ACE emulator format), only with snapshot version 3+
- -ok <breakpoint filename>: set the full filename for breakpoint export.
- -eb : Export breakpoints in a text file
- -sb : Export breakpoints in snapshot file (Winape and ACE emulator format), only with snapshot version 3+

7.5 CPC+ Colors

7.5.1 GET_R, GET_G, GET_B

```
GET_R <16 bits RGB value>
GET_G <16 bits RGB value>
GET_B <16 bits RGB value>
```

Use this in an expression, in order to get one of the 4 bit component of a 16 bit color as used in the ASIC

7.5.2 SET_R, SET_G, SET_B

```
SET_R <4 bits value>
SET_G <4 bits value>
SET_B <4 bits value>
```

Returns a 16 bit value where the 4-bit value of the color component (R,G,B) is set

```
| dw (SET_R 4) | (SET_G 15) | (SET_B 0) ; Defines RGB Color (4,15,0)
```

you also can define your own macro like this:

```
macro drgb dr,db,dg
dw SET_R dr | SET_G dg | SET_B db
mend
```

7.6 Deprecated Directives

7.6.1 NOCODE

```
NOCODE
...
CODE
```

This directive is used for disabling code generation for a portion of code.

7.6.2 WRITE DIRECT

```
WRITE DIRECT <lower rom>[,<higher rom>[,<RAM gate array>]]
```

This directive is only supported for Winape compatibility. Prefer usage of `BANK` or `BANKSET` directives.

7.6.3 LIST, NOLIST, LET

These directives are ignored. Usage for Maxam/Winape compatibility only.

8 ZX Specific features

RASM also features a few options and directives specific for ZX architecture.

8.1 HOBETA Directive

`HOBETA`

Use this directive for generating a file in HOBETA format

8.2 Bank Selection

8.2.1 BUILDZX Directive

`BUILDZX <bank>`

Use this directive for selecting a bank (0..7)

8.3 Options

`-sx` option can be used for exporting symbols for ZX emulators, where the selected bank is output. `(<bank>:<adresse>)`

9 Compiling and Embedding

9.1 Building RASM

There is no installation procedure for RASM, as it consists in a single executable file. As C source code is provided with RASM, it can be recompiled. Here are the commands for some platforms and compilers - mainly calling scripts or Makefiles All these scripts use UPX, a compression tool for executable binaries, available here: https://upx.github.io

Linux

```
#Makefile invocation
make prod
```

Windows (Visual Studio)

```
combil.bat
```

Dos/Windows 32 (Watcom) For DosBox, you need to set RAM to at least 64Mb

```
msdos.bat
```

MacOS

```
make makefile.MacOS
```

9.2 Embedding RASM

There are 3 steps t follow for integrating RASM into your own C/C++ application:

- Build RASM as an object binary that can be linked. In order to do this, it must be compiled with `INTEGRATED_ASSEMBLY` symbol.
- Include 'rasm.h' in our program, and use one of the two functions for assembling our Z80 code.
- Building our application, without forgetting to link the binary file produced in step 1.

As a first example, we'll use `RasmAssemble` function, which returns an error code (0 if everything went fine, or -1 if an error happened), and also assembled byte code, in an array of unsigned chars. `RasmAssembleInfos` is similar, but it also returns additional data, such as a list of errors and the value of all symbols.

```
#include "rasm.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Prgram to assemble
const char* prog="org #9000 \n\
    ld b,10 \n\
lp: djnz lp \n\
    ret \n";

int main(void) {
  printf("%s\n", prog);
```

```
  unsigned char *buf = NULL;
  int asmsize = 0;
  int res = RasmAssemble(prog, strlen(prog), &buf, &asmsize);

  printf("Result=%d, Generated code size=%d\n",res,asmsize);

  if (res==0)
  {
      for (int i=0; i<asmsize; i++)
      {
        printf("%02X ", buf[i]);
        if ((i&15)==15) printf("\n");
      }
      printf("\n");
    }
  else
    {
      printf("Failure!\n");
    }

  if (buf)
    free(buf);

  return 0;
}
```

In order to copile rasm and our example (embed.cpp):

```
gcc -D INTEGRATED_ASSEMBLY rasm_v0111.c -c -o rasm_embedded.obj
gcc embed.cpp rasm_embedded.obj -lm
```

When executed, it produces this:

```
$ ./a.out
org #9000
ld b,10
lp:  djnz lp
ret
Result=0, Generated code size=5
06 0A 10 FE C9
```

9.2.1 Errors and Symbols

A Syntactic coloration

A.1 Syntax color with VIM

If you already have a syntax color file, just add the following lines to the file .vim/syntax/z80.vim Or you may download the whole file here

```
" rasm/winape directives
syn keyword z80PreProc charset bank write save include incbin incl48 incl49
syn keyword z80PreProc macro mend switch case break while wend repeat until
syn keyword z80PreProc buildcpr amsdos lz48 lz49 lzclose protect
syn keyword z80PreProc direct brk let print stop nolist str
syn keyword z80PreProc defr dr defi undef
syn keyword z80PreProc bankset page pageset sizeof endm struct endstruct ends
syn keyword z80PreProc incexo lzexo lzx7 inczx7 buildsna setcrtc setcpc assert print
syn keyword z80Reg lix liy hix hiy
```

B Z80 Opcodes

B.1 Main Instructions

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	nop	ld bc,**	ld (bc),a	inc bc	inc b	dec b	ld b,*	rlca	ex af,af'	add hl,bc	ld a,(bc)	dec bc	inc c	dec c	ld c,*	rrca
1	djnz *	ld de,**	ld (de),a	inc de	inc d	dec d	ld d,*	rla	jr *	add hl,de	ld a,(de)	dec de	inc e	dec e	ld e,*	rra
2	jr nz,*	ld hl,**	ld (**),hl	inc hl	inc h	dec h	ld h,*	daa	jr z,*	add hl,hl	ld hl,(**)	dec hl	inc l	dec l	ld l,*	cpl
3	jr nc,*	ld sp,**	ld (**),a	inc sp	inc (hl)	dec (hl)	ld (hl),*	scf	jr c,*	add hl,sp	ld a,(**)	dec sp	inc a	dec a	ld a,*	ccf
4	ld b,b	ld b,c	ld b,d	ld b,e	ld b,h	ld b,l	ld b,(hl)	ld b,a	ld c,b	ld c,c	ld c,d	ld c,e	ld c,h	ld c,l	ld c,(hl)	ld c,a
5	ld d,b	ld d,c	ld d,d	ld d,e	ld d,h	ld d,l	ld d,(hl)	ld d,a	ld e,b	ld e,c	ld e,d	ld e,e	ld e,h	ld e,l	ld e,(hl)	ld e,a
6	ld h,b	ld h,c	ld h,d	ld h,e	ld h,h	ld h,l	ld h,(hl)	ld h,a	ld l,b	ld l,c	ld l,d	ld l,e	ld l,h	ld l,l	ld l,(hl)	ld l,a
7	ld (hl),b	ld (hl),c	ld (hl),d	ld (hl),e	ld (hl),h	ld (hl),l	halt	ld (hl),a	ld a,b	ld a,c	ld a,d	ld a,e	ld a,h	ld a,l	ld a,(hl)	ld a,a
8	add a,b	add a,c	add a,d	add a,e	add a,h	add a,l	add a,(hl)	add a,a	adc a,b	adc a,c	adc a,d	adc a,e	adc a,h	adc a,l	adc a,(hl)	adc a,a
9	sub b	sub c	sub d	sub e	sub h	sub l	sub (hl)	sub a	sbc a,b	sbc a,c	sbc a,d	sbc a,e	sbc a,h	sbc a,l	sbc a,(hl)	sbc a,a
A	and b	and c	and d	and e	and h	and l	and (hl)	and a	xor b	xor c	xor d	xor e	xor h	xor l	xor (hl)	xor a
B	or b	or c	or d	or e	or h	or l	or (hl)	or a	cp b	cp c	cp d	cp e	cp h	cp l	cp (hl)	cp a
C	ret nz	pop bc	jp nz,**	jp **	call nz,**	push bc	add a,*	rst #00	ret z	ret	jp z,**	prefix CB	call z,**	call **	adc a,*	rst #08
D	ret nc	pop de	jp nc,**	out (*),a	call nc,**	push de	sub *	rst #10	ret c	exx	jp c,**	in a,(*)	call c,**	prefix DD	sbc a,*	rst #18
E	ret po	pop hl	jp po,**	ex (sp),hl	call po,**	push hl	and *	rst #20	ret pe	jp (hl)	jp pe,**	ex de,hl	call pe,**	prefix ED	xor *	rst #28
F	ret p	pop af	jp p,**	di	call p,**	push af	or *	rst #30	ret m	ld sp,hl	jp m,**	ei	call m,**	prefix FD	cp *	rst #38

B.2 Extended instructions (ED)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
4	in b,(c)	out (c),b	sbc hl,bc	ld (**),bc	neg	retn	im 0	ld i,a	in c,(c)	out (c),c	adc hl,bc	ld bc,(**)	neg	reti	im 0/1	ld r,a
5	in d,(c)	out (c),d	sbc hl,de	ld (**),de	neg	retn	im 1	ld a,i	in e,(c)	out (c),e	adc hl,de	ld de,(**)	neg	retn	im 2	ld a,r
6	in h,(c)	out (c),h	sbc hl,hl	ld (**),hl	neg	retn	im 0	rrd	in l,(c)	out (c),l	adc hl,hl	ld hl,(**)	neg	retn	im 0/1	rld
7	in (c)	out (c),0	sbc hl,sp	ld (**),sp	neg	retn	im 1		in a,(c)	out (c),a	adc hl,sp	ld sp,(**)	neg	retn	im 2	
A	ldi	cpi	ini	outi					ldd	cpd	ind	outd				
B	ldir	cpir	inir	otir					lddr	cpdr	indr	otdr				

B.3 Bit instructions (CB)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	rlc b	rlc c	rlc d	rlc e	rlc h	rlc l	rlc (hl)	rlc a	rrc b	rrc c	rrc d	rrc e	rrc h	rrc l	rrc (hl)	rrc a
1	rl b	rl c	rl d	rl e	rl h	rl l	rl (hl)	rl a	rr b	rr c	rr d	rr e	rr h	rr l	rr (hl)	rr a
2	sla b	sla c	sla d	sla e	sla h	sla l	sla (hl)	sla a	sra b	sra c	sra d	sra e	sra h	sra l	sra (hl)	sra a
3	sll b	sll c	sll d	sll e	sll h	sll l	sll (hl)	sll a	srl b	srl c	srl d	srl e	srl h	srl l	srl (hl)	srl a
4	bit 0,b	bit 0,c	bit 0,d	bit 0,e	bit 0,h	bit 0,l	bit 0,(hl)	bit 0,a	bit 1,b	bit 1,c	bit 1,d	bit 1,e	bit 1,h	bit 1,l	bit 1,(hl)	bit 1,a
5	bit 2,b	bit 2,c	bit 2,d	bit 2,e	bit 2,h	bit 2,l	bit 2,(hl)	bit 2,a	bit 3,b	bit 3,c	bit 3,d	bit 3,e	bit 3,h	bit 3,l	bit 3,(hl)	bit 3,a
6	bit 4,b	bit 4,c	bit 4,d	bit 4,e	bit 4,h	bit 4,l	bit 4,(hl)	bit 4,a	bit 5,b	bit 5,c	bit 5,d	bit 5,e	bit 5,h	bit 5,l	bit 5,(hl)	bit 5,a
7	bit 6,b	bit 6,c	bit 6,d	bit 6,e	bit 6,h	bit 6,l	bit 6,(hl)	bit 6,a	bit 7,b	bit 7,c	bit 7,d	bit 7,e	bit 7,h	bit 7,l	bit 7,(hl)	bit 7,a
8	res 0,b	res 0,c	res 0,d	res 0,e	res 0,h	res 0,l	res 0,(hl)	res 0,a	res 1,b	res 1,c	res 1,d	res 1,e	res 1,h	res 1,l	res 1,(hl)	res 1,a
9	res 2,b	res 2,c	res 2,d	res 2,e	res 2,h	res 2,l	res 2,(hl)	res 2,a	res 3,b	res 3,c	res 3,d	res 3,e	res 3,h	res 3,l	res 3,(hl)	res 3,a
A	res 4,b	res 4,c	res 4,d	res 4,e	res 4,h	res 4,l	res 4,(hl)	res 4,a	res 5,b	res 5,c	res 5,d	res 5,e	res 5,h	res 5,l	res 5,(hl)	res 5,a
B	res 6,b	res 6,c	res 6,d	res 6,e	res 6,h	res 6,l	res 6,(hl)	res 6,a	res 7,b	res 7,c	res 7,d	res 7,e	res 7,h	res 7,l	res 7,(hl)	res 7,a
C	set 0,b	set 0,c	set 0,d	set 0,e	set 0,h	set 0,l	set 0,(hl)	set 0,a	set 1,b	set 1,c	set 1,d	set 1,e	set 1,h	set 1,l	set 1,(hl)	set 1,a
D	set 2,b	set 2,c	set 2,d	set 2,e	set 2,h	set 2,l	set 2,(hl)	set 2,a	set 3,b	set 3,c	set 3,d	set 3,e	set 3,h	set 3,l	set 3,(hl)	set 3,a
E	set 4,b	set 4,c	set 4,d	set 4,e	set 4,h	set 4,l	set 4,(hl)	set 4,a	set 5,b	set 5,c	set 5,d	set 5,e	set 5,h	set 5,l	set 5,(hl)	set 5,a
F	set 6,b	set 6,c	set 6,d	set 6,e	set 6,h	set 6,l	set 6,(hl)	set 6,a	set 7,b	set 7,c	set 7,d	set 7,e	set 7,h	set 7,l	set 7,(hl)	set 7,a

B.4 IX instructions (DD)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0										add ix,bc						
1										add ix,de						
2		ld ix,**	ld (**),ix	inc ix	inc ixh	dec ixh	ld ixh,*			add ix,ix	ld ix,(**)	dec ix	inc ixl	dec ixl	ld ixl,*	
3					inc (ix+*)	dec (ix+*)	ld (ix+*),*			add ix,sp						
4					ld b,ixh	ld b,ixl	ld b, (ix+*)						ld c,ixh	ld c,ixl	ld c, (ix+*)	
5					ld d,ixh	ld d,ixl	ld d, (ix+*)						ld e,ixh	ld e,ixl	ld e, (ix+*)	
6	ld ixh,b	ld ixh,c	ld ixh,d	ld ixh,e	ld ixh,ixh	ld ixh,ixl	ld h, (ix+*)	ld ixh,a	ld ixl,b	ld ixl,c	ld ixl,d	ld ixl,e	ld ixl,ixh	ld ixl,ixl	ld l, (ix+*)	ld ixl,a
7	ld (ix+*),b	ld (ix+*),c	ld (ix+*),d	ld (ix+*),e	ld (ix+*),h	ld (ix+*),l		ld (ix+*),a					ld a,ixh	ld a,ixl	ld a, (ix+*)	
8					add a,ixh	add a,ixl	add a, (ix+*)						adc a,ixh	adc a,ixl	adc a, (ix+*)	
9					sub ixh	sub ixl	sub (ix+*)						sbc a,ixh	sbc a,ixl	sbc a, (ix+*)	
A					and ixh	and ixl	and (ix+*)						xor ixh	xor ixl	xor (ix+*)	
B					or ixh	or ixl	or (ix+*)						cp ixh	cp ixl	cp (ix+*)	
C												prefix DDCB				
D																
E		pop ix		ex (sp),ix		push ix				jp (ix)						
F										ld sp,ix						

B.5 IX bit instructions (DDCB)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	rlc (ix+*),b	rlc (ix+*),c	rlc (ix+*),d	rlc (ix+*),e	rlc (ix+*),h	rlc (ix+*),l	rlc (ix+*)	rlc (ix+*),a	rrc (ix+*),b	rrc (ix+*),c	rrc (ix+*),d	rrc (ix+*),e	rrc (ix+*),h	rrc (ix+*),l	rrc (ix+*)	rrc (ix+*),a
1	rl (ix+*),b	rl (ix+*),c	rl (ix+*),d	rl (ix+*),e	rl (ix+*),h	rl (ix+*),l	rl (ix+*)	rl (ix+*),a	rr (ix+*),b	rr (ix+*),c	rr (ix+*),d	rr (ix+*),e	rr (ix+*),h	rr (ix+*),l	rr (ix+*)	rr (ix+*),a
2	sla (ix+*),b	sla (ix+*),c	sla (ix+*),d	sla (ix+*),e	sla (ix+*),h	sla (ix+*),l	sla (ix+*)	sla (ix+*),a	sra (ix+*),b	sra (ix+*),c	sra (ix+*),d	sra (ix+*),e	sra (ix+*),h	sra (ix+*),l	sra (ix+*)	sra (ix+*),a
3	sll (ix+*),b	sll (ix+*),c	sll (ix+*),d	sll (ix+*),e	sll (ix+*),h	sll (ix+*),l	sll (ix+*)	sll (ix+*),a	srl (ix+*),b	srl (ix+*),c	srl (ix+*),d	srl (ix+*),e	srl (ix+*),h	srl (ix+*),l	srl (ix+*)	srl (ix+*),a
4	bit 0, (ix+*)	bit 0, (ix+*)	bit 0, (ix+*)	bit 0, (ix+*)	bit 0, (ix+*)	bit 0, (ix+*)	bit 0, (ix+*)	bit 0, (ix+*)	bit 1, (ix+*)	bit 1, (ix+*)	bit 1, (ix+*)	bit 1, (ix+*)	bit 1, (ix+*)	bit 1, (ix+*)	bit 1, (ix+*)	bit 1, (ix+*)
5	bit 2, (ix+*)	bit 2, (ix+*)	bit 2, (ix+*)	bit 2, (ix+*)	bit 2, (ix+*)	bit 2, (ix+*)	bit 2, (ix+*)	bit 2, (ix+*)	bit 3, (ix+*)	bit 3, (ix+*)	bit 3, (ix+*)	bit 3, (ix+*)	bit 3, (ix+*)	bit 3, (ix+*)	bit 3, (ix+*)	bit 3, (ix+*)
6	bit 4, (ix+*)	bit 4, (ix+*)	bit 4, (ix+*)	bit 4, (ix+*)	bit 4, (ix+*)	bit 4, (ix+*)	bit 4, (ix+*)	bit 4, (ix+*)	bit 5, (ix+*)	bit 5, (ix+*)	bit 5, (ix+*)	bit 5, (ix+*)	bit 5, (ix+*)	bit 5, (ix+*)	bit 5, (ix+*)	bit 5, (ix+*)
7	bit 6, (ix+*)	bit 6, (ix+*)	bit 6, (ix+*)	bit 6, (ix+*)	bit 6, (ix+*)	bit 6, (ix+*)	bit 6, (ix+*)	bit 6, (ix+*)	bit 7, (ix+*)	bit 7, (ix+*)	bit 7, (ix+*)	bit 7, (ix+*)	bit 7, (ix+*)	bit 7, (ix+*)	bit 7, (ix+*)	bit 7, (ix+*)
8	res 0, (ix+*),b	res 0, (ix+*),c	res 0, (ix+*),d	res 0, (ix+*),e	res 0, (ix+*),h	res 0, (ix+*),l	res 0, (ix+*)	res 0, (ix+*),a	res 1, (ix+*),b	res 1, (ix+*),c	res 1, (ix+*),d	res 1, (ix+*),e	res 1, (ix+*),h	res 1, (ix+*),l	res 1, (ix+*)	res 1, (ix+*),a
9	res 2, (ix+*),b	res 2, (ix+*),c	res 2, (ix+*),d	res 2, (ix+*),e	res 2, (ix+*),h	res 2, (ix+*),l	res 2, (ix+*)	res 2, (ix+*),a	res 3, (ix+*),b	res 3, (ix+*),c	res 3, (ix+*),d	res 3, (ix+*),e	res 3, (ix+*),h	res 3, (ix+*),l	res 3, (ix+*)	res 3, (ix+*),a
A	res 4, (ix+*),b	res 4, (ix+*),c	res 4, (ix+*),d	res 4, (ix+*),e	res 4, (ix+*),h	res 4, (ix+*),l	res 4, (ix+*)	res 4, (ix+*),a	res 5, (ix+*),b	res 5, (ix+*),c	res 5, (ix+*),d	res 5, (ix+*),e	res 5, (ix+*),h	res 5, (ix+*),l	res 5, (ix+*)	res 5, (ix+*),a
B	res 6, (ix+*),b	res 6, (ix+*),c	res 6, (ix+*),d	res 6, (ix+*),e	res 6, (ix+*),h	res 6, (ix+*),l	res 6, (ix+*)	res 6, (ix+*),a	res 7, (ix+*),b	res 7, (ix+*),c	res 7, (ix+*),d	res 7, (ix+*),e	res 7, (ix+*),h	res 7, (ix+*),l	res 7, (ix+*)	res 7, (ix+*),a
C	set 0, (ix+*),b	set 0, (ix+*),c	set 0, (ix+*),d	set 0, (ix+*),e	set 0, (ix+*),h	set 0, (ix+*),l	set 0, (ix+*)	set 0, (ix+*),a	set 1, (ix+*),b	set 1, (ix+*),c	set 1, (ix+*),d	set 1, (ix+*),e	set 1, (ix+*),h	set 1, (ix+*),l	set 1, (ix+*)	set 1, (ix+*),a
D	set 2, (ix+*),b	set 2, (ix+*),c	set 2, (ix+*),d	set 2, (ix+*),e	set 2, (ix+*),h	set 2, (ix+*),l	set 2, (ix+*)	set 2, (ix+*),a	set 3, (ix+*),b	set 3, (ix+*),c	set 3, (ix+*),d	set 3, (ix+*),e	set 3, (ix+*),h	set 3, (ix+*),l	set 3, (ix+*)	set 3, (ix+*),a
E	set 4, (ix+*),b	set 4, (ix+*),c	set 4, (ix+*),d	set 4, (ix+*),e	set 4, (ix+*),h	set 4, (ix+*),l	set 4, (ix+*)	set 4, (ix+*),a	set 5, (ix+*),b	set 5, (ix+*),c	set 5, (ix+*),d	set 5, (ix+*),e	set 5, (ix+*),h	set 5, (ix+*),l	set 5, (ix+*)	set 5, (ix+*),a
F	set 6, (ix+*),b	set 6, (ix+*),c	set 6, (ix+*),d	set 6, (ix+*),e	set 6, (ix+*),h	set 6, (ix+*),l	set 6, (ix+*)	set 6, (ix+*),a	set 7, (ix+*),b	set 7, (ix+*),c	set 7, (ix+*),d	set 7, (ix+*),e	set 7, (ix+*),h	set 7, (ix+*),l	set 7, (ix+*)	set 7, (ix+*),a

B.6 IY instructions (FD)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0										add iy,bc						
1										add iy,de						
2		ld iy,**	ld (**),iy	inc iy	inc iyh	dec iyh	ld iyh,*			add iy,iy	ld iy,(**)	dec iy	inc iyl	dec iyl	ld iyl,*	
3					inc (iy+*)	dec (iy+*)	ld (iy+*),*			add iy,sp						
4					ld b,iyh	ld b,iyl	ld b, (iy+*)						ld c,iyh	ld c,iyl	ld c, (iy+*)	
5					ld d,iyh	ld d,iyl	ld d, (iy+*)						ld e,iyh	ld e,iyl	ld e, (iy+*)	
6	ld iyh,b	ld iyh,c	ld iyh,d	ld iyh,e	ld iyh,iyh	ld iyh,iyl	ld h, (iy+*)	ld iyh,a	ld iyl,b	ld iyl,c	ld iyl,d	ld iyl,e	ld iyl,iyh	ld iyl,iyl	ld l, (iy+*)	ld iyl,a
7	ld (iy+*),b	ld (iy+*),c	ld (iy+*),d	ld (iy+*),e	ld (iy+*),h	ld (iy+*),l		ld (iy+*),a					ld a,iyh	ld a,iyl	ld a, (iy+*)	
8					add a,iyh	add a,iyl	add a, (iy+*)						adc a,iyh	adc a,iyl	adc a, (iy+*)	
9					sub iyh	sub iyl	sub (iy+*)						sbc a,iyh	sbc a,iyl	sbc a, (iy+*)	
A					and iyh	and iyl	and (iy+*)						xor iyh	xor iyl	xor (iy+*)	
B					or iyh	or iyl	or (iy+*)						cp iyh	cp iyl	cp (iy+*)	
C												prefix FDCB				
D																
E		pop iy		ex (sp),iy		push iy				jp (iy)						
F										ld sp,iy						

B.7 IY bit instructions (FDCB)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	rlc (iy+*),b	rlc (iy+*),c	rlc (iy+*),d	rlc (iy+*),e	rlc (iy+*),h	rlc (iy+*),l	rlc (iy+*)	rlc (iy+*),a	rrc (iy+*),b	rrc (iy+*),c	rrc (iy+*),d	rrc (iy+*),e	rrc (iy+*),h	rrc (iy+*),l	rrc (iy+*)	rrc (iy+*),a
1	rl (iy+*),b	rl (iy+*),c	rl (iy+*),d	rl (iy+*),e	rl (iy+*),h	rl (iy+*),l	rl (iy+*)	rl (iy+*),a	rr (iy+*),b	rr (iy+*),c	rr (iy+*),d	rr (iy+*),e	rr (iy+*),h	rr (iy+*),l	rr (iy+*)	rr (iy+*),a
2	sla (iy+*),b	sla (iy+*),c	sla (iy+*),d	sla (iy+*),e	sla (iy+*),h	sla (iy+*),l	sla (iy+*)	sla (iy+*),a	sra (iy+*),b	sra (iy+*),c	sra (iy+*),d	sra (iy+*),e	sra (iy+*),h	sra (iy+*),l	sra (iy+*)	sra (iy+*),a
3	sll (iy+*),b	sll (iy+*),c	sll (iy+*),d	sll (iy+*),e	sll (iy+*),h	sll (iy+*),l	sll (iy+*)	sll (iy+*),a	srl (iy+*),b	srl (iy+*),c	srl (iy+*),d	srl (iy+*),e	srl (iy+*),h	srl (iy+*),l	srl (iy+*)	srl (iy+*),a
4	bit 0, (iy+*)	bit 0, (iy+*)	bit 0, (iy+*)	bit 0, (iy+*)	bit 0, (iy+*)	bit 0, (iy+*)	bit 0, (iy+*)	bit 0, (iy+*)	bit 1, (iy+*)	bit 1, (iy+*)	bit 1, (iy+*)	bit 1, (iy+*)	bit 1, (iy+*)	bit 1, (iy+*)	bit 1, (iy+*)	bit 1, (iy+*)
5	bit 2, (iy+*)	bit 2, (iy+*)	bit 2, (iy+*)	bit 2, (iy+*)	bit 2, (iy+*)	bit 2, (iy+*)	bit 2, (iy+*)	bit 2, (iy+*)	bit 3, (iy+*)	bit 3, (iy+*)	bit 3, (iy+*)	bit 3, (iy+*)	bit 3, (iy+*)	bit 3, (iy+*)	bit 3, (iy+*)	bit 3, (iy+*)
6	bit 4, (iy+*)	bit 4, (iy+*)	bit 4, (iy+*)	bit 4, (iy+*)	bit 4, (iy+*)	bit 4, (iy+*)	bit 4, (iy+*)	bit 4, (iy+*)	bit 5, (iy+*)	bit 5, (iy+*)	bit 5, (iy+*)	bit 5, (iy+*)	bit 5, (iy+*)	bit 5, (iy+*)	bit 5, (iy+*)	bit 5, (iy+*)
7	bit 6, (iy+*)	bit 6, (iy+*)	bit 6, (iy+*)	bit 6, (iy+*)	bit 6, (iy+*)	bit 6, (iy+*)	bit 6, (iy+*)	bit 6, (iy+*)	bit 7, (iy+*)	bit 7, (iy+*)	bit 7, (iy+*)	bit 7, (iy+*)	bit 7, (iy+*)	bit 7, (iy+*)	bit 7, (iy+*)	bit 7, (iy+*)
8	res 0, (iy+*),b	res 0, (iy+*),c	res 0, (iy+*),d	res 0, (iy+*),e	res 0, (iy+*),h	res 0, (iy+*),l	res 0, (iy+*)	res 0, (iy+*),a	res 1, (iy+*),b	res 1, (iy+*),c	res 1, (iy+*),d	res 1, (iy+*),e	res 1, (iy+*),h	res 1, (iy+*),l	res 1, (iy+*)	res 1, (iy+*),a
9	res 2, (iy+*),b	res 2, (iy+*),c	res 2, (iy+*),d	res 2, (iy+*),e	res 2, (iy+*),h	res 2, (iy+*),l	res 2, (iy+*)	res 2, (iy+*),a	res 3, (iy+*),b	res 3, (iy+*),c	res 3, (iy+*),d	res 3, (iy+*),e	res 3, (iy+*),h	res 3, (iy+*),l	res 3, (iy+*)	res 3, (iy+*),a
A	res 4, (iy+*),b	res 4, (iy+*),c	res 4, (iy+*),d	res 4, (iy+*),e	res 4, (iy+*),h	res 4, (iy+*),l	res 4, (iy+*)	res 4, (iy+*),a	res 5, (iy+*),b	res 5, (iy+*),c	res 5, (iy+*),d	res 5, (iy+*),e	res 5, (iy+*),h	res 5, (iy+*),l	res 5, (iy+*)	res 5, (iy+*),a
B	res 6, (iy+*),b	res 6, (iy+*),c	res 6, (iy+*),d	res 6, (iy+*),e	res 6, (iy+*),h	res 6, (iy+*),l	res 6, (iy+*)	res 6, (iy+*),a	res 7, (iy+*),b	res 7, (iy+*),c	res 7, (iy+*),d	res 7, (iy+*),e	res 7, (iy+*),h	res 7, (iy+*),l	res 7, (iy+*)	res 7, (iy+*),a
C	set 0, (iy+*),b	set 0, (iy+*),c	set 0, (iy+*),d	set 0, (iy+*),e	set 0, (iy+*),h	set 0, (iy+*),l	set 0, (iy+*)	set 0, (iy+*),a	set 1, (iy+*),b	set 1, (iy+*),c	set 1, (iy+*),d	set 1, (iy+*),e	set 1, (iy+*),h	set 1, (iy+*),l	set 1, (iy+*)	set 1, (iy+*),a
D	set 2, (iy+*),b	set 2, (iy+*),c	set 2, (iy+*),d	set 2, (iy+*),e	set 2, (iy+*),h	set 2, (iy+*),l	set 2, (iy+*)	set 2, (iy+*),a	set 3, (iy+*),b	set 3, (iy+*),c	set 3, (iy+*),d	set 3, (iy+*),e	set 3, (iy+*),h	set 3, (iy+*),l	set 3, (iy+*)	set 3, (iy+*),a
E	set 4, (iy+*),b	set 4, (iy+*),c	set 4, (iy+*),d	set 4, (iy+*),e	set 4, (iy+*),h	set 4, (iy+*),l	set 4, (iy+*)	set 4, (iy+*),a	set 5, (iy+*),b	set 5, (iy+*),c	set 5, (iy+*),d	set 5, (iy+*),e	set 5, (iy+*),h	set 5, (iy+*),l	set 5, (iy+*)	set 5, (iy+*),a
F	set 6, (iy+*),b	set 6, (iy+*),c	set 6, (iy+*),d	set 6, (iy+*),e	set 6, (iy+*),h	set 6, (iy+*),l	set 6, (iy+*)	set 6, (iy+*),a	set 7, (iy+*),b	set 7, (iy+*),c	set 7, (iy+*),d	set 7, (iy+*),e	set 7, (iy+*),h	set 7, (iy+*),l	set 7, (iy+*)	set 7, (iy+*),a

C Z80 Opcodes Duration on CPC

This table shows the duration of all z80 opcodes, expressed in number of equivalent NOPs. This is valid for CPC only. For example, ADD A,(HL) has the same duration as 2 NOPs.

- r: 8 bits register (A,B,C,D,E,H,L)
- rr: 16 bits register (AF,BC,DE,HL,SP)
- d: 8 bits data (0..255)
- dd: 16 bits data
- b: bit (0..7)
- cond: Condition (CC,Z,M,NC,NZ,P,PO,PE)

Instruction using IY register have exactly the same duration as IX instructions, they are omitted in the table.

Opcode	Duration
ADC r,r	1
ADC A,(HL)	2
ADC A,n	2
ADC HL,rr	4
ADC HL,SP	4
ADC A,IXH	2
ADC A,IXL	2
ADC A,(IX+d)	5
ADD r,r	1
ADD A,(HL)	2
ADD A,d	2
ADD HL,dd	3
ADD IX,rr	4
ADD IX,IX	4
ADD IX,SP	4
ADD A,IXH	2
ADD A,IXL	2
ADD A,(IX+d)	5
AND r	1
AND d	2
AND (HL)	2
AND IXH	2
AND IXL	2
AND (IX+d)	5
BIT r	2
BIT (HL)	3
BIT b,(IX+d)	6
BIT b,(IX+d),r	6
CALL dd	5
CALL cond,dd	5 / 3
CCF	1
CP r	1
CP d	2
CP (HL)	2
CP IXH	2
CP IXL	2
CP (IX+d)	3
CPD	4
CPDR	6 / 4
CPI	4
CPIR	6 / 4
CPL	1
DAA	1
DEC r	1
DEC rr	2
DEC (HL)	3
DEC IX	3
DEC IXH	2
DEC IXL	2
DEC (IX+d)	6
DI	1
DJNZ d	4 / 3
EI	1
EX AF,AF'	1
EX DE,HL	1
EX (SP),HL	6
EX (SP),IX	7
EXX	1
HALT	1
IM 0	2
IM 1	2
IM 2	2
IN A,(d)	3
IN r,(C)	4
INC r	1
INC rr	2
INC (HL)	3
INC IX	3
INC IXH	2
INC IXL	2
INC (IX+d)	6
IND	5
INI	5
INIR	6 / 5
INDR	6 / 5
JP dd	3
JP cond,dd	3
JP (HL)	1
JP (IX)	2
JR d	3
JR C,d	3 / 2
JR NC,d	3 / 2
JR NZ,d	3 / 2
JR Z,d	3 / 2
LD r,r	1
LD r,d	2
LD A,(rr)	2
LD r,(HL)	2
LD (rr),A	2
LD SP,HL	2
LD r,IXH	2
LD r,IXL	2
LD SP,IX	3
LD rr,dd	3
LD (HL),d	3
LD A,R	3
LD R,A	3
LD A,I	3
LD I,A	3
LD IXH,d	3
LD IXL,d	3
LD A,(dd)	4
LD (dd),A	4
LD IX,dd	4
LD HL,(dd)	5

Opcode	Duration
LD BC,(dd)	6
LD DE,(dd)	6
LD (dd),HL	5
LD r,(IX+d)	5
LD (dd),rr	6
LD IX,(dd)	6
LD (dd),IX	6
LD (IX+d),r	5
LD (IX+d),d	6
LD (dd),SP	6
LD SP,(dd)	6
LDD	5
LDI	5
LDDR	6 / 5
LDIR	6 / 5
NEG	2
NOP	1
OR r	1
OR d	2
OR (HL)	2
OR IXH	2
OR IXL	2
OR (IX+d)	5
OUT (d),A	3
OUT (C),r	4
OUT (C),0	4
OUTD	5
OUTI	5
OTDR	6 / 5
OTIR	6 / 5
POP rr	3
POP IX	4
PUSH rr	4
PUSH IX	5
RES b,r	2
RES b, (HL)	4
RES b,(IX+d)	7
RES b,(IX+d),r	7

Opcode	Duration
RET	3
RET cond	4 / 2
RETN	4
RETI	4
RL r	2
RL (HL)	4
RL (IX+d)	7
RL (IX+d),r	7
RLC r	2
RLC (HL)	4
RLC (IX+d)	7
RLC (IX+d),r	7
RLCA	1
RLA	1
RLD	5
RR r	2
RR (HL)	4
RR (IX+d)	7
RR (IX+d),r	7
RRA	1
RRC r	2
RRC (HL)	4
RRC (IX+d)	7
RRC (IX+d),r	7
RRD	5
RRCA	1
RST d	4
SBC A,r	1
SBC A,d	2
SBC A,IXH	2
SBC A,IXL	2
SBC A,(HL)	2
SBC A,(IX+d)	5
SBC HL,rr	4
SBC HL,SP	4

Opcode	Duration
SCF	1
SET b,r	2
SET b, (HL)	4
SET b,(IX+d)	7
SET b,(IX+d),r	7
SLA r	2
SLA (HL)	4
SLA (IX+d)	7
SLA (IX+d),r	7
SLL r	2
SLL (HL)	4
SLL (IX+d)	7
SLL (IX+d),r	7
SRA r	2
SRA (HL)	4
SRA (IX+d)	7
SRA (IX+d),r	7
SRL r	2
SRL (HL)	4
SRL (IX+d)	7
SRL (IX+d),r	7
SUB r	1
SUB d	2
SUB (HL)	2
SUB IXH	2
SUB IXL	2
SUB (IX+d)	5
XOR r	1
XOR d	2
XOR (HL)	2
XOR IXH	2
XOR IXL	2
XOR (IX+d)	5

Index