RASM

roudoudou Assembler
v0.75

Documentation RASM v0.75 (c) Édouard BERGÉ 2017

Table of Contents

Introduction

18 years ago, I programmed an assembler/disassembler called Zasm/dizasm. In fact it was mainly a proof of concept of a mono-pass assembler. I used a little the disassembler with personnal projects. Sources were published but the spread was so confidential that it was hard to retrieve them back on my HDD.

Since then, another assembler called Zasm exists, created by a spectrum developper team.

As the code of Rasm is from scratch (except a few concept), it's the pretext to choose a new name: Rasm.

Installation

Rasm a stand-alone executable, there is no installation.

Compilation

Linux compilation:

```
cc rasm_v075.c -O2 -lm -lrt -march=native
mv a.out rasm
strip rasm
```

Windows compilation with Visual Studio:

```
cl.exe rasm_v075.c -O2
```

Morphos compilation:

```
ppc-morphos-gcc-5 -O2 -c -o rasm rasm_v075.c
strip rasm
```

Rasm behaviour

Rasm is supposed to be easy to use. It will set default filename, guess what memory region to save or create a cartridge file is ROM bank were selected during assembly.

Rasm allow multiple ORG in the same memory set but it will not allow to write twice the same adress in the same memory set. Each ORG, Rasm check there is no overwrite.

If you need to generate many code portion at the same adress, you have two possibilities. Either you use the <output> parameter of ORG directive (to choose another output adress), or you may create another memory set using directive WRITE DIRECT -1,-1,#C0

When you select ROM with directives BANK or WRITE DIRECT, this is always the same memory set for each ROM. If you need another memory set, do not use ROM.

When using SAVE directive, Rasm disable automatic binary output to file or cartridge. You may want to force CPR writing with directive BUILDCPR.

When possible, Rasm try to display comprehensive error messages to point out the problem.

Rasm is using a pre-processor to normalize code, remove useless spaces, comment, check quotes and legal characters, and finally transform some instructions into others. As an example, XOR,AND,OR,MOD will be converted in a single char like C syntax.

When a read file directive do not refer to an absolute path, le root path is the one from the current file.

Using command line

The mandatory syntaxe is: rasm.exe <file to assemble> [options]

Example: `rasm.exe myfile.asm -s`

File option

-o <output prefix> set the base name for default output files (binary, cartridge, symbols). The default value is “rasmoutput”.

`rasm.exe src.asm -o output -s` will produce the following filenames: `output.bin, output.cpr, output.sym`

-ob <binary filename> set the full name for binary file output
-oc <cartridge filename> set the full name for cartridge file output
-oi <snapshot filename> set the full name for snapshot file output
-os <symbol filename> set the full name for symbol file output
-ok <breakpoint filename>set the full name for breakpoints file output
-no disable file output

EDSK output

-eo overwrite files on floppy image if there is a name collision

Symbol option

-s export symbols to Rasm format
-sp export symbols to Pasmo format
-sw export symbols to Winape format
-ss export symbols in the snapshot
-eb export breakpoints

-sl additionnal option to the three previous one. It exports also local symbols.
-sv other additionnal option which exports variables.
-sq other additionnal option which exports alias EQU.
-sa option meaning -sl -sv -sq

- Arnold emulator can handle any export format
- Winape cannot handle label of hiw own format when they are too long. The debuger will crash.

-l <label file> import symbols before assembling

Compatibility option

-m maxam calculation style

- comparison operator is a single =
- all calculations are done with unsigned integer 16 bits values and wrong rounding

-ass AS80 behaviour mimic

- all calculations are done with 32 bits integer values and wrong rounding
- macro parameters are not protected with {} anymore
- multiple declarations with DEFB,DEFW,DEFI got reference adress of the first outputed byte. That's why with AS80, using multiple DEFB produces a different result than using DEFB with multiple values
- MACRO declaration directive must be used after the macro name

Developper options

- -v simple verbose mode
- -d pre-processing verbose mode
- -a assembling verbose mode
- -n display third-parties licences

Code source format

generalities

Assembleur is not COBOL, it is useless to indent sources with Rasm, except to be pretty. You may use two points to separete labels, or not. Rasm try to be free style. The direct consequence of this free writing style involves a difference with the assemblers of obsolete design. It is not possible (fortunately) to create a label that has the same name as a directive or Z80 instruction.

Files may be read to Unix or Windows format, an internal conversion is done.

Rasm is not case sensible. All chars will be converted upper case. Don't be surprise when Rasm display error messages.

Comments

Rasm comment starts with semicolon. All chars are ignored until next carriage return.

There is no multiline comment.

Literal values

Rasm knows the following literals:

- In decimal if the value begin with a digit.
- In binary if the value begin with a % or 0b.
- In octal if the value begin with a @.
- In hexadecimal if the value begin with a #, a $, 0x or ends with h.
- An ascii value if a single character is between quote.
- An internal value (variable or label) if the literal begin with a letter or an '@' for local labels.
- $ symbol used alone means *starting adress of the current instruction*. When using a DEFB, DEFW or DEFL, the current adress is those from current element. A DEF* instruction outputs the same thing if you use multiple args or multiple DEF*.

Rasm does all its internal calculations in double floating-point precision. A correct rounding is performed at the end of the calculation chain for the integer needs.

Caution, the & character is reserved for the AND operator.

Authorized characters

Between quotations, all characters are allowed, at your own risk, for ASCII conversion to Amstrad. outside quotes, you can use all letters, all digits, point, arobas, parentheses, dollar, plus, minus, multiplied, divided, pipe, circumflex, percent, sharp, the paragraph, the rafters and the two types of quotes.

Files

Rasm automatically convert path into Unix style internally. As a consequence, if you use only relative paths (with filename that windows can read), the code may be compiled with Rasm Unix and Rasm Windows without modification.

The file management rule is quite simple. Each relative path root iare located to the root of the file where it was read.

Labels

Inside a loop (REPEAT / WHILE/UNTIL) or inside a macro it is possible to use local labels in the same way as with the integrated Winape assembler prefixing the label with the character '@'.

For each loop iteration, for each loop nesting, a suffix is added to the local label containing the hexadecimal value of the internal repetition counter. It is thus possible to call a local label to a repetition outside the loop, but this use is not advised.

It's possible to declare a label starting by a dot. This is an old declaration style. The call to this label must be done without the starting dot.

It's possible to use a label in a directive (ORG for example) if the label declaration is before the directive.

Specific tags for labels

You can use the prefix {BANK} with a label (example: {BANK}mylabel) to get the number of the bank where the label is located, instead of the label adress.

You can use the prefix {PAGE} with a label (example: {PAGE}mylabel) to get the gate array banking value for the label, instead of the label adress. Example for a label located in the BANK 5 → #C4

You can use the prefix {PAGESET} with a label (example: {PAGESET}mylabel) to get the gate array banking value for the 64K set of a label, instead of the label adress. Example for a label located in the BANK 5 → #C2

Expressions

Rasm use a simple expression engine with priorities (C-style) how can handle the following operators/functions:

Compute operands

- * multiplication
- / division
- + addition
- - substraction
- & logical operator AND
- | logical operator OR
- ^ logical operator exclusive OR
- && boolean operator AND
- || boolean operator OR
- % or § modulo (use MOD if your keyboard does not support this char)
- << multiply by a power of 2 (left shifting)
- >> divide by a power of 2 (right shifting)
- AND, OR, XOR, MOD inline maxam style
- sin() sinus
- cos() cosinus
- asin() arc-sinus
- acos() arc-cosinus
- atan() arc-tangente
- int() double to int conversion
- floor() truncate value to lower integer
- abs() absolute value
- ln() natural logarithm
- log10() logarithm base 10
- exp() exponential
- sqrt() square root

Comparison operators:

- == equality (only one = in maxam compatibility mode)
- != different from
- <= less than or equal to
- >= greater than or equal to
- < less than
- > greater than

Warning about expressions

~~Comparison operators musn't be included inside parenthesis:~~
~~right syntax: `IF (4+myvar)*5==myresult`~~

~~wrong syntax: `IF ((4+myvar)*5==myresult)`~~

Static variables or alias

It's possible to create aliases with EQU directive. Thoses aliases can't be modified.

Dynamic variables

Rasm may use an unlimited number of variables for internal calculations.

Syntax: `mavariable=5` or `LET mavariable=5`

Thoses variables may be used as loop counter, offset or value in a loop or the main code.

Example:

```
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
```

Directives

A directive is not a function, this is a keyword which must be hhh from his parameters with at least one space char.

Right syntax: `ASSERT (4*myvar)`
Wrong syntax: `ASSERT(4*myvar)`

ASSERT <condition>

Check condition and stop assembling if result is not true

EQU

Create an alias

Example:
```
myvar EQU 5
myothervar EQU myvar*2
```

ALIGN <boundary>

Align code to a given boundary

Examples:
`ALIGN 2` to align to even adress
`ALIGN 256` to align to the high byte adress

AMSDOS

Add an Amsdos header to the automatic binary file. Note: There is no Amsdos header added with SAVE directive.

BREAKPOINT <adress>

Add a breakpoint (this is not a Z80 instruction. No byte is outputed). Breakpoints may be exported after assembling in a raw file or in snapshots (ACE and Winape emulators are supported).

Any label beginning with BRK or @BRK will produce a label and a breakpoint.

You may set an optionnal adress to force the breakpoint anywhere in the memory.

BUILDCPR

Force cartridge output when using SAVE directive.

BUILDSNA <V2>

Select snapshot output (snapshot v3). The default machine is a 6128 with CRTC 0 but you may change the target with SETCRTC and SETCPC directives.

The snapshot is initialised with the Basic screen size, all ink deep blue except pen 1 with bright yellow. All 3 audio channels are disabled, rom disabled and IM 1.

Example: create a snapshot using multiple memory selection

```
buildsna ; should be declared before using bank to avoid 512K limit

bankset 0 ; assemble in first 64K of the CPC
org #1000
run #1000 ; snapshot will start at #1000

ld b,#7F
ld a,{page}mydata ; get gate array style bank number of the label
out (c),c
ld a,(mydata)
jr $

bank 6 ; select 3rd bank of 2nd 64K set
nop
mydata defb #DD

bank ; without parameter, create an independant memory space that
; won't be saved in the snapshot

pouet
repeat 10
cpi
rend
camion

save"anotherthing",pouet,camion-pouet
```

Compatibility option in order to create snapshot v2:

Some emulators or hardware card does not handle snapshot v3. To downgrade snapshot to v2 version (128K maximum, not compressed), just add V2 to the directive.

```
buildsna v2
```

BANK [<rom page number>,<ram page number>]

Select a ROM for assembling. Even if a ROM size is 16K, you can start to write anywhere in the ROM. Rasm will use the first outputed byte as start adress when creating CPR file. ROM number range is from 0 to 31 and RAM number range is from 0 to 35.

Without parameter, the directive opens a new memory workspace like the ugly WRITE DIRECT -1,-1,#C0

BANKSET <64k block number>

Select 4 pages at a time. This option implies snapshot generation. The bankset 0 stand for pages 0,1,2,3 then the bankset 1 stand for pages 4,5,6,7 …

If you mix BANK and BANKSET, do not use pages of a bankset. A control mechanism will check you are not trying to do so.

BANKSET usage enable snapshot generation.

IF, IFNOT, ELSE, ELSEIF, ENDIF

Conditionnal assembly

Example:

```
CODE_PRODUCTION=1
[...]
IF CODE_PRODUCTION
OR #80
ELSE
PRINT 'Test version'
ENDIF
```

IFDEF, IFNDEF <variable>

Both directives will test if variable exists or not.

LZ48 / LZ49 / LZ4 / LZX7 / LZEXO

Open a LZ48, LZ49, LZ4, ZX7 or Exomizer crunched code part. The output code will be crunched after assembly and following code will be relocated.

It's not possible to call a label located after the crunched code, from the crunched code as we cannot know how big will be the crunched code. There will be an error if you try so.

Limitations:

- It's not possible to assemble a code bigger than 64ko before crunch.
- It's not possible to embed portions of crunched code.

LZCLOSE

Close a crunched part opened with LZ48, LZ49, LZ4, LZX7 or LZEXO

READ / INCLUDE 'file to read'

Read a text file and include it into the current source code. Relative path starts from the absolute location of the current file. An absolute path do not care about this.

There is not recursion limit. Be aware of what you do.

INCBIN 'file to read'[,offset[,size[,extended offset[,OFF]]]]

Include a binary file. Read data will be directly outputed. Parameters are like Winape syntax except first offset is not limited to 64K. Extended offset is here for compatibility only.

The offset parameter may be a negative value. Then the offset is relative to the end of the file.

The size parameter may be a negative value. Then this value is relative to the filesize, in order to read less bytes. If you want to stop 10 bytes before the end, juste put -10 in the filesize.

A zero size will load all the file.

OFF parameter: If you want to load a file in order to initialise memory then assembling code over it, you may disable the overwrite check.

Example:

```
org #4000
incbin'makeraw.bin',0,0,0,OFF ; read the file in #4000
org #4001
defb #BB ; overwrite second byte of the file without error
```

INCL48 / INCL49 / INCLZ4 / INCZX7 / INCEXO 'file to read'

Read a binary file, crunch it with LZ48, LZ49, LZ4, LZX7 or Exomizer and directly output in the memory.

LIMIT <maximum adress>

Set a lower limit (other than 64K) for outputed code. If you want to protect a zone, you may use PROTECT directive.

ORG <code adress>[,<output adress>]

Assemble at a given adress. You may set an output adress to generate the code at a different memory adress.

PROTECT <start adress>,<end adress>

Protect memory region in the current memory set. Any write to this region will trigger an error and a message.

STR '<string>','<string2>',...

Similar to DEFB '<string>', the last character of each string has bit 7 set to 1 (OR #80 on the last byte of each string).

STOP

Stop assembling.

PRINT '<string>',variables,expressions,...

Write text/variables/formulas during assembling. It's possible to set display format with tag prefixes.

`{hex}` Display in hexa. If the value is less than #100 then the display is forced to 2 digits. If the value is less than #10000 then the display is forced to 4 digits. With upper value there won't be any extra-zeros.

`{hex2}, {hex4}, {hex8}` display is forced to 2, 4 or 8 digits, with any value.

`{bin}` display value as binary. If the value is less than #100 then the display is forced to 8 bits. If the value is less than #10000 then the display is forced to 16 bits. With upper value there won't be any extra-zeros. A preprocessing remove the 16 upper bits of the 32 bits value if all bits are set to one (aka 16 bits negative value).

`{bin8},{bin16},{bin32}` display is forced to 8, 16 or 32 digits, with any value.

`{int}` display rounded integer value.

Without prefix the value is displayed as a floating-point value.

LIST / NOLIST / LET

Directives are ignored, only for compatibility with Winape.

RUN <adress>

Set the run adress when using snapshot output.

WHILE / WEND

Repeat a block of instructions everytime the condition is true. For each loop you may use a variable while_counter to trigger actions regarding of the iteration counter.

Example:

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

The loop will run 10 times, producing 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)
```

REPEAT <n> / UNTIL | REPEAT / UNTIL <condition>

Repeat a block of instructions. You may set a fixed number of repetition or set a conditionnal repeat. As the WHILE directive, you may use an internal iteration counter named repeat_counter.

Examples:

```
repeat 16
ldi
print repeat_counter
rend

cpt=10
repeat
ldi
cpt=cpt-1
until cpt>0
```

WRITE DIRECT <lower rom>,<upper rom>,<RAM>

If you set a ROM number (lower from 0 to 7, upper from 0 to 31), the directive will have the very same effect as BANK directive.

If you set the RAM (disabling ROM with -1 both lower and upper ROM), Rasm create a new memory set for assembling. Using multiple WRITE DIRECT you can assemble many code at the same adress, but different memory sets.

Example:

```
;default memory space, ORG at zero
defs 65536,0
WRITE DIRECT -1,-1,#C0
defs 65536,0
; new memory space, no matter the RAM bank
; in this example twice #C0
WRITE DIRECT -1,-1,#C0
; no error since it's a third memory space
defs 65536,0
```

Rasm created three memory spaces. The default memory space and two additionnal set for each WRITE DIRECT call. There is no limit for memory space, except your system memory.

When a new memory space is created, you cannot get back to any previous one.

SAVE 'binary file to write',<adress>,<size>

Write a binary file starting from adress of the current memory set to adress+size. Note that the saves are done only if there is no error at the end of assembly.

Write a binary file

```
SAVE 'myfilename.bin',start,size
```

Write a binary file with AMSDOS header

```
SAVE 'myfilename.bin',start,size,AMSDOS
```

Write a binary file on floppy image

```
SAVE 'myfile.bin',start,size,DSK,'myedsk.dsk'
```

The filename will be automatically truncated and set in upper case if it does not fit the AMSDOS specs.

SETCPC <modèle>

Choose CPC model for snapshot generation. Possible values are:

- 0 : 464
- 1 : 664
- 2 : 6128
- 4 : 464+
- 5 : 6128+
- 6 : GX-4000

SETCRTC <modèle>

Choose CRTC model for snapshot generation. Values from 0 to 4.

CHARSET 'string',<value> | <byte>,<value> | <start>,<end>,<value>

Allows the value of characters or characters in strings to be assigned alternative values. There are 4 forms of this directive.

- 'string',<value> First character of the string will have a new ascii <value>. Next char will have <value>+1 etc. until the end of the string.
- <byte>,<value> set a new <value> for the character <byte>
- <start>,<end>,<value> Set from the character <start> until <end> an incremental value starting from <value>.
- “no parameter” will set default values to all characters.

This directive is compatible with Winape.

SWITCH, CASE, BREAK, DEFAULT, ENDSWITCH

The syntax of those directives is like C syntax except you can write more than one case with the same value, which gives more flexibility.

In this example, calling the macro with 5 will assemble all 'yes' and 'yes again' strings:

```
macro yesyes myvar
switch {myvar}
    nop ; won't be assembled because outside case
    case 3
        defb 'no'
    case 5
        defb 'yes'
    case 7
        defb 'again yes'
        break
    case 8
        defb 'no'
    case 5
        defb 'again yes'
        break
    default
        defb 'no'
endswitch
mend
```

Macros

Rasm handle macros with curly brackets (Winape compatible). It is possible to do conditionnal assembling inside macros because each macro call insert a brand new code with new substitutions of parameters. Then the code is interpreted like if there were no macro.

Example for a long distance LD (IX+offset),register write:

```
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
```

Note: It's possible to use ENDM as MEND alias

Macros without parameter

If you are using a macro without parameter, I encourage you to add the useful (VOID) parameter. Then, if you misspell the macro, it won't be understand by Rasm as a new label.

```
macro withoutparam
nop
mend

withoutparam (void) ; secured macro call
```

Calling macro with static or dynamic parameters

Rasm macros are in fact a dynamic code replacement in the source when assembling. You may have two parameters replacement modes: The parameter is copied as is, or it may be computed once for all then the result will be setted.

```
macro test myarg
defb {myarg}
mend

repeat 2
test repeat_counter
rend
repeat 2
test {eval}repeat_counter
rend
```

In the first loop, the line `defb repeat_counter` will be copied whereas in the second call, with prefix taf `{eval}`, only the first value of repeat_counter will be copied, producing two identical `defb`.

Undocumented instructions

All documented and undocumented instructions are supported.

Index registers IX and IY 8-bits form may be used with ixl, lx ou xl, etc.

Complex instructions syntax:

```
res 0,(ix+0),a
bit 0,(ix+0),a
sll 0,(ix+0),a
rl  0,(ix+0),a
rr  0,(ix+0),a
```

Input/Output undocumented instructions syntax:

```
out (<n>),a ; <n> is a 8 bits value
in a,(<n>)
```

`in 0,(c)` or `in f,(c)`

Authorized syntaxes:

```
push bc,de,hl ; → push bc : push de : push hl
nop 4         ; → nop : nop : nop : nop
```

Limitations

- It's not possible to use instruction or directive as label.
- It's not possible to declare label,alias or variable with the same name.
- Rasm is not case sensitive.
- Crunched section cannot exceed 64K before crunch.

RASM programmation examples

Retrieve the bank of a label in order to connect a ROM

Using the {BANK} prefix allow to move label in another bank without changing the code.

```
Bank 0
ld a,{bank}maroutine ; will assemble LD A,1
call connect_bank    ; define your own!
jp maroutine

bank 1
defb 'coucou'
maroutine
jr $
```

Create a snapshot

```
Buildsna
bankset 0 ; assemble in the full first 64K
org #1000
; program
nop

bank 4 ; assemble ni the first page of extended memory of a 6128
org #4000
defb 1
; we will connect the bank in #4000 si it's easy to org in #4000
; but the page is still 16K as we use BANK directive
```

Saving a binary on floppy image DSK

Rasm is able to create or modify floppy images DSK or EDSK

```
org #8000

startcode

ld hl,hello
printcharloop
ld a,(hl)
or a
ret z
call #BB5A
inc hl
jr printcharloop
hello defb 'Hello world',13,10,0

endcode

save"hello.bin",startcode,endcode-startcode,DSK,"hello.dsk"
```