CASM

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

A simple, portable multi-pass assembler

Chapter 1

Usage

```
casm file
```

Assembles file, and places the output in *output* by default.

Chapter 2

Source Format Example

The source files follow this basic format:

```
; Comments
;
label1: equ     0xffff

        org     $4000;

        db      "Hello, World\n",0

main    jp      local_label     ; Comments

.local_label
        inc     a

another:
        inc     b
        jp      local_label     ; Actually jumps to the following local_label.

.local_label
        ret
```

The source files follow the following rules:

- Any text past a semicolon (;) is discarded as a comment (except when part of a string constant).
- Labels must start in column zero (the left hand most column).
  - If the label ends with a colon (:) then the colon is removed.
  - If the label doesn't start with a period (.) then it is assumed a global label.
  - If the label starts with a period (.) then it is assumed to be a local label. Local labels are associated with the preceding global label. If a global label and related local label have the same name, the local label will be used on expansion.
  - Any label can be followed by an *equ* directive, in which case the label is set to that value rather than the current program counter.
  - Labels are case-insensitive.
- Directives and opcodes must appear further along the line (anywhere else other than the left hand column where labels live basically).
- Strings can either be quoted with single or double quotes; this allows you to put the other quote type inside the string.

2.1 Recognised directives

All directives are also recognised with an optional period (.) in front of them, and are case insensitive. Directives can also be used to control the output of a program listing, and the output of the assembly itself. These are documented in further sections.

processor *CPU*
Sets the processor type to *CPU*. If omitted then Z80 is the default. Note that this can appear multiple times in the same file. Currently supported *CPU* values are `Z80` and `6502`.

option *setting*, *value*
Set options. Options are defined later on, and each CPU can also have its own options. For options that support booleans (on/off/true/false), the *setting* can be prefixed with a plus or minus character to switch it on or off respectively.

equ *value*
Sets the top level label to *value*. Note this requires a label on the same line.

org *value*
Sets the program counter (PC) to *value*. The PC defaults to zero.

ds *value*[, *fill*]
Skips on the program counter *value* bytes. If the optional *fill* is provided then the bytes are filled with *fill*, otherwise they are filled with zero.

db *value*[, *value*]
Writes bytes to the current PC. The values can be constants, expressions, labels or strings. Built-in aliases are `byte` and `text`.

dw <value>[, <value>]
Writes words (16-bit values) to the current PC. The values can be constants, expressions or labels. Note that `word` is a built-in alias for this directive.

align *value*[, *fill*]
Align the PC so that (PC modulus *value*) is zero. Will error if *value* is less than 2 or greater that 32768. No values are written to the skipped bytes unless the optional *fill* is supplied.

include *filename*
Includes the source file *filename* as if it was text entered at the current location.

incbin *filename*
Includes the binary data in *filename* at the current PC, as if it was a sequence of `db` directives with all the bytes from the file.

alias *command*, *replacement*
Creates an alias so that whenever the command *command* is found in the source it is replaced with *replacement*. The idea of this is to make it easier to import sources that use unknown directives, e.g.

```
alias setaddr,org
alias ldreg,ld

cpu         z80

setaddr     $8000   ; These two are
org         $8000   ; equivalent.

ld          a,(hl)  ; These two are
ldreg       a,(hl)  ; equivalent.
```

nullcmd
Simply does nothing. It's only real use is as an alias if you wished to strip a directive from a foreign source file.

end
Terminates the input processing. Anything past the directive will be ignored.

2.2 Expressions

In any of the directives above, where a value is defined, an expression can be entered.

The following formats for constant numbers are supported (note these are illustrated as a regular expression):

"x" or *x*
: A single quoted character will be converted into the appropriate character code.

[1-9][0-9]*
: A decimal number, e.g. 42.

0[0-7]*
: An octal number, e.g. 052.

0x[0-9a-fA-f]+
: A hex number, e.g. 0x2a.

[0-9a-fA-f]+h
: A hex number, e.g. 2ah.

$[0-9a-fA-f]+
: A hex number, e.g. $2a.

[01]+b
: A binary number, e.g. 00101010b

[a-zA-Z_0-9]+
: A label, e.g. `main_loop`.

The following operators are understood. The order here is the order of precedence.

{ }
: Brackets used to alter the order of precedence. Note normal parenthesis aren't used as the assembly language may make use of them.

~ + -
: Bitwise NOT/unary plus/unary minus.

<< >>
: Shift left/shift right.

/ * %
: Division/multiplication/modulus.

+ -
: Addition/subtraction.

All the following have the same precedence, and so will be done left to right.

==
: Equality. Returns 1 if the arguments are equal, otherwise zero.

!=
: Inequality. Returns 1 if the arguments are unequal, otherwise zero.

< <= > >=
: Less than/less than or equal/greater than/greater than or equal. Returns 1 if the arguments are equal, otherwise zero.

All the following have the same precedence, and so will be done left to right.

&& &
: Boolean/bitwise AND. For boolean operation arguments, zero is FALSE, otherwise TRUE.

|| |
: Boolean/bitwise OR.

^
: Bitwise XOR.

Assembly instructions will also permit these expressions to be used where applicable. As many opcodes use parenthesis to indicate addressing modes, remember that { } brackets can be used to alter expression precedence.

```
    ld  a,{8+2}*2               ; On the Z80 loads A with the value 20
    ld  a,({8+2}*2)             ; On the Z80 loads A with the value stored at
                                ; address 20
```

Note that the expression is evaluated using a standard C int, and then cast to the appropriate size.

2.3 Character Sets

The assembler has built-in support for a few different character sets. These can be set by using the options *charset* or *codepage*, i.e.

```
    option codepage, <format>
    option charset, <format>
```

The following values can be used for *format*.

ascii
: 7-bit ASCII. This is the default.

spectrum
: The character codes as used on the Sinclair ZX Spectrum.

zx81
: The character codes as used on the Sinclair ZX-81. Lower case letters are encoded as normal upper case letters and upper case letter will be encoded as inverse upper case letters.

cbm
: PETSCII as used on the Commodore Business Machine's range from the PET to the C128. See https://en.wikipedia.org/-wiki/PETSCII for more details.

e.g.

```
    option  +list
    option  +list-hex

    option  charset,ascii
    db      "Hello",'A'
; $48 $65 $6C $6C $6F $41

    option  charset,zx81
    db      "Hello",'A'
; $AD $2A $31 $31 $34 $A6

    option  codepage,cbm
    db      "Hello",'A'
; $48 $45 $4C $4C $4F $41

    option  codepage,spectrum
    db      "Hello",'A'
; $48 $65 $6C $6C $6F $41
```

2.4 Macros

Macros can be defined in one of two ways; either parameterless or with named parameters. Macro names are case-insensitive. In the parameterless mode the special identifier * can be used to expand all arguments, which will be separated with commas.

```
macro1: macro

        ld a,\1
        ld b,\2
        call \3
        defb \*

        endm

macro2: macro char,junk,interface

        ld a,@char
        ld b,@junk
        call @interface

        endm
```

Note that trying to expand and unknown/missing argument will be replaced with an empty string. Also the two argument reference styles can be mixed, though obviously the @ form only makes sense in a parameterised macro, e.g.

```
mac:    macro char,junk,interface

        ld a,@char
        ld b,\2
        call @interface

        endm
```

The at symbol (@) used for parameter expansion in named argument macros can be replaced by using the following option, e.g.

```
        option  macro-arg-char,&
```

Note that this is enforced when the macro is **used**, not when it is **defined**. Also the character must not be quoted, as that will be parsed as a string holding the character code of the character.

Chapter 3

Output Format

By default the assembled code is written to a file called **output** as raw binary covering the block of memory that the assembly touched.

This can be controlled with the following options.

option output-file, ***file***
Send the output to *file*.

option output-type, ***format***
Controls the output format with the following settings

raw
The default raw binary.

spectrum
Generates a Spectrum TAP file for an emulator. The TAP file will be given the same name as the output filename, and its load address will be set to the start of the created memory. Remember that TAP files can be concatenated, so the output could be appended to another TAP file containing a BASIC loader for example.

Chapter 4

Listing

By default no output listing is generated. This can be controlled by the the following options.

option list, <on|off>
Enables/disables listing. The listing will go to stdout.

option list-file, ***file***
Sends the listing to *file*. Note this should appear before enabling the listing.

option list-pc, <on|off>
Control the output of the current PC in the as a comment preceding the line (so that a listing could be reassembled with no editing). Defaults to **off**.

option list-hex, <on|off>
Control the output of the bytes generated by the source line in hex. Defaults to **off**. If **on** then the hex is output in a comment preceding the line (possibly with the PC above), so that a listing is still valid to be assembled.

option list-labels, <on|off|all>
Controls the listing of labels, either **off** (the default), **on** to dump label values at the end of the listing and **all** to dump all labels, including internally generated private labels for macros.

option list-macros, <off|exec|dump|all>
Controls the listing of macro invocations, either

off
The default; don't list anything.

exec
List invocations of macros.

dump
Produce a list of macro definitions at the end of the listing.

all
Combine "exec" and "dump"

option list-rm-blanks, <on|off>
Defaults to **on**. This option causes multiple blank lines to be collapsed down to a single line.

Chapter 5

Z80 CPU

5.1 Opcodes

The Z80 assembler uses the standard Zilog opcodes, and supports undocumented instructions.

For instructions were the Accumulator can be assumed it can be omitted, and EOR can be used the same as XOR:

```
xor     a,a         ; These are equivalent
xor     a
eor     a,a

and     a,b         ; These are equivalent
and     b
```

For exchange opcodes with parameters the parameters can be reversed from their official form:

```
; The official forms
;
ex      de,hl
ex      af,af'
ex      (sp),hl
ex      (sp),ix
ex      (sp),iy

; Also supported
;
ex      hl,de
ex      af',af
ex      hl,(sp)
ex      ix,(sp)
ex      iy,(sp)
```

Where the high/low register parts of the IX and IY registers are to be used, simply use ixl, iyl, ixh and iyh. Note that the assembler will accept illegal pairings involving H and L, but these will be warned about:

```
ld  ixh,$e5
ld  iyl,iyl

ld  ixh,l           ; This will be turned into "ld ixh,ixl" and a
                    ; warning will be issued.

ld  iyh,ixl         ; This will generate an error as the index registers
                    ; have been mixed.
```

For bit manipulations that also can copied to a register, these can be represented by adding the destination register as an extra parameter, e.g.

```
srl (iy-1),d
set 3,(iy-1),a
res 4,(iy-1),b
```

For the hidden IN instruction using the flag register the following are all equivalent:

```
in  (c)
in  f,(c)
```

For the hidden OUT instruction using the flag register, $00 or $ff depending on where you're reading, the following are all equivalent, where *value* can be any value at all:

```
out (c)
out (c),f
out (c),<value>
```

5.2 Options

The Z80 assembler has no options.

Chapter 6

6502 CPU

6.1 Opcodes

The 6502 assembler uses the standard Motorola opcodes.

6.2 Options

The 6502 assembler has the following options.

option zero-page, <on|off|auto>
Use Zero-Page addressing for *absolute* and *absolute*,X address modes. If mode is set to **auto** then tries to calculate the mode based on the value in the last pass. Defaults to **off**. e.g.

```
cpu     6502
org     $8000

lda     $0000,x     ; Produces $bd $00 $00
option  +zero-page
lda     $0000,x     ; Produces $b5 $00
lda     $1234,x     ; Produces an error

option  zero-page,auto
lda     $00,x       ; Produces $b5 $00
lda     $8000,x     ; Produces $bd $00 $80
```