Minis assembler 0.10

Programmer's Manual

Table of Contents :

This document may contain errors.

1 - What is Minis assembler :

Minis assembler is a small assembler for processor intel 80386
or better, and support almost all instruction of this cpu, this
assembler generate 16 or 32 code in one or multiple passes,
and by default create flat binary file.
This assembler is used to compile minis operating system.

2 - How to use this assembler :

This assembler compatible with MSDOS 2.0 or better and executed only from command line of MSDOS or any DOS emulators like DosBox.
You can download and install DosBox on many operating systems
(windows/linux/android ...).
After installing DosBox check documentation of how to mount folders
than mount folder that contain your assember & source file and type :
as [source file name] [binary file name]
for example :

```
as boot.asm boot.bin
```

if no error found in source file you will see size of binary file and
the output from cmd look like this:
success: 14 bytes.
This message mean the size of binary file is 14 bytes.

If assembler find error durring the compilation process, you will see error message.
for example:
error: invalid label.
line : [5]
This message mean the error is invalid label at line number 5, and of course at source file.
to fix this error you must follow rules at chapter (3.6- Define labels).

error: unknown instruction.
line : [13]
This message mean at line number 13 the instruction is not supported by minis assembler or not exist in assembly language.
If you are beginner, check assembly programming tutorials.

3 - Assembler syntax :
Minis assembler follow intel syntax .

3.1 - Number syntax :
Numbers can be written in Hexadecimal, decimal or binary.
Hexadicimal must be in UPPERCASE and from 0x00 to 0xFFFFFFFF
or you get message : "error: invalid operand."
for example :
0xF3, 0x23FA, 0x0016, 0xFFFFFFFF ; Hexadecimal
10, 54563, 456, 97753689 ; Decimal
10110011b, 00000100b, 1011b ; Binary

3.2 - Instruction syntax :
Minis assembler accept only one instruction at each line
followed by line break or comment.

registers :
reg8 : al, cl, dl, bl, ah, ch, dh, bh
reg16 : ax, cx, dx, bx, sp, bp, si, di
reg32 : eax, ecx, edx, ebx, esp, ebp, esi, edi
index : eax, ecx, edx, ebx, esi, edi, bx, si, di
seg : cs, ss, ds, es, fs, gs
cr : cr0, cr2, cr3

ib : immediate byte (1-byte)
imm8 : immediate byte (1-byte)
imm16 : immediate word (2-byte)
imm32 : immediate dword (4-byte)
m8 : memory byte (1-byte)
m16 : memory word (2-byte)
m32 : memory dword (4-byte)

for example :
push es
int 0xF3
mov byte [si], 0x34 ; put immediate value 0x34 into the byte at address (content of register 'si')
mov eax, here ; put address of label (here) into register eax
mov edx, 0x12 ; put immediate value 0x12 into register edx
mov ss, ax ; mov must have same size of registers
mov ebx, eax ; mov must have same size of registers

3.3 - Directives :
use16 : set code type 16-bit (real mode)
use32 : set code type 32-bit (protected mode)
times : repeat instruction number of times (max=65535).
binary : include binary file inside source code.

for example :
times 0x05 db 0x00 ; repeat define (byte 0x00) five times.
binary 'icon.bmp'

3.4 - Comments :
Any lines start or followed by semicolon (;) is comment lines and the assembler will ignores from semicolon to the end of line during assembling the source code, except to the semicolon (;) found between quote (').
for example :
; I am just comment
int 0xF3 ; call to interrupt 0xF3

3.5 - Define data :
To define data use one of this directives (db,dw,dd) followed by one value or more separated with commas, also labels can be used as value (offset of label).
for example :
db 0x00 ; single byte
db 'hello world', 0x00 ; sequence of bytes
dw 0x1015 ; define 2 byte
dd 0xFFFFFFFF ; define 4 byte
this_address:
db 0x33, this_address, 0x25 ; label inside define sequence of bytes
pointer_n01:
dd this_address ; example of creating pointer

3.6 - Define labels :
Label must be alone in line not followed by instruction, and not start with number.
To define label you must use characters from (a) to (z) in lowercase, numbers and underscore (_), labels must be between 1 to 32 characters, and followed by colon (:).
for example :
label_number_12:
start:
_jmp_here:

3.7 - Jumps :

Unconditional jump :

for example :

```
        jmp here
        nop
        here:
jmp 0x1000: word 0x0000  ; 16-bit far jump
jmp 0x8: dword 0x10050    ; 32-bit far jump
jmp bx      ; jmp at address in bx register
jmp eax     ; jmp at address in eax register
jmp word [bx]    ; jmp to memory found at address in bx register
jmp dword [eax]  ; jmp to memory found at address in eax register
jmp word [here]  ; jmp to memory at offset 'here'
jmp dword [25678]  ; jmp to memory address '25678'
```

Conditional jump :

for example :

```
        jc there
        je there
        nop
        there:
```

3.8 - Calls :

for example :

```
call this_label ; run code at label 'this_label'
; and return here after call instruction
this_label:
mov si, 10
ret
call bx   ; call address in bx register
call eax  ; call address in eax register
call 0x1000: word 0x0000  ; 16-bit far call
call 0x8: dword 0x10050    ; 32-bit far call
call word [there]  ; call near word memory indirect
call dword [there] ; call near dword memory indirect
call word [ebx]    ; call near word register indirect
```

3.9 - Move data :
The instruction 'mov' used to copy content from the source operand to the destination operand, with operands of the same size (byte,word,dword).
Data movement work like this:
mov destination, source

Minis assembler support only transfer between registers or from immediate value to register, or from immediate value to memory or index registers and imposible from memory to memory.
The operators:
'+,-' can be used with memory, register or number.
'*' can be used only in scale index base addressing

imm8/imm16/imm32 can be number, label or characters between quote ''.
the instruction 'mov reg8, byte [index]' move data at index register (eax, ecx, edx, ebx, esi, edi, bx, si, di) to 8-bit register (al, cl, dl, bl, ah, ch, dh, bh).
whene index register used alone the defaut segment is data segment (ds), you can also use segment register with index register.

for example :

```
mov dl, byte [si]      ; this copy one byte of memory at address ds:si to dl register.
mov ax, word [es:bx]   ; this copy two bytes of memory at address es:bx to ax register.
mov al, 'A'            ; this put character A into al register.
mov eax, dword [esi+1]
mov byte [there], 0x0F ; store immediate byte '0x0f' to memory at address 'there'
mov al, byte [there]   ; copy content of memory at address 'there' to al register.
mov ah, byte [there+1] ; copy content of memory at address 'there' + 1 to ah register.
mov byte [ebx+esi*2], 13 ; scale index base addressing.
```

To copy data to or from a register use the following rules:

```
mov reg8, reg8
mov reg8, imm8
mov reg8, byte [index/m8]
mov byte [index/m8], reg8
mov byte [index/m8], imm8

mov reg16, reg16
mov reg16, imm16
mov reg16, word [index/m16]
mov word [index/m16], reg16
mov word [index/m16], imm16
```

```
mov reg32, reg32
mov reg32, imm32
mov reg32, dword [index/m32]
mov dword [index/m32], reg32
mov dword [index/m32], imm32

mov seg, reg16
mov reg16, seg
mov cr, reg32
mov reg32, cr
```

3.10 - System Instruction :
The instruction 'lgdt' used to load value in operand into the global descriptor table register.
for example :

```
gdt_r:
        dw 0x17     ; limit (Size of GDT)
        dd 0x1001E  ; base of GDT
             ; pword = 6 bytes

        mov eax, gdt_r
        lgdt pword [eax]   ; load GDT into GDTR
        lgdt pword [gdt_r] ; sime as above
        sgdt pword [tmp_memory] ; copies the contents of the GDT register to 6 bytes memory 'tmp_memory'
```

The instruction 'lidt' used to load value in operand into the interrupt descriptor table register.
for example :

```
idt_r:
        dw 0xFE     ; limit (Size of IDT)
        dd 0x100F8  ; base of IDT
             ; pword = 6 bytes
        mov eax, idt_r
        lidt pword [eax]    ; load IDT into IDTR
        lidt pword [idtr_r] ; sime as above
        sidt pword [tmp_memory]   ; copies the contents of the IDT register to 6 bytes memory 'tmp_memory'
```

4 - instruction supported :

aaa
aad
aam
aas
adc
add
and
arpl
bound
bsf/bsr
bt/btc/btr/bts
call
cbw/cwde
clc
cld
cli
clts
cmc
cmp
cmps/cmpsb/cmpsw/cmpsd
cwd/cdq
daa
das
dec
div
enter
hlt
idiv
imul
in
inc
insb/insw/insd
int
int3
into
iret/iretd
ja/jae/jb/jbe/jc/jcxz/jecxz/je/jz/jg/jge/jl/jle/jna/jnae/jnb/jnbe/
jnc/jne/jng/jnge/jnl/jnle/jno/jnp/jns/jnz/jo/jp/jpe/jpo/js/
jmp
lahf
lar
lea

leave
lgdt/lidt
lgs/lss/lds/les/lfs
lldt
lmsw
lock
lods/lodsb/lodsw/lodsd
loop
lsl
ltr
mov
movs/movsb/movsw/movsd
movsx
movzx
mul
neg
nop
not
or
out
outsb/outsw/outsd
pop
popa/popad
popf
push
pusha/pushad
pushf
rcl/rcr/rol/ror
rep/repe/repz/repne/repnz
ret
sahf
sal/sar/shl/shr
sbb
scasb/scasw/scasd
setcc
sgdt/sidt
sldt
smsw
stc
std
sti
stos/stosb/stosw/stosd
str
sub

test
verr/verw
wait
xlatb
xor