Minis assembler 0.09a

Programmer's Manual

Table of contents :

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 with just two 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 executed only from command line of windows operating
system (Minimum Windows XP), and need only 1 MB of Ram memory.
open cmd 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 like this:

```
success: 14 bytes.
```

This message mean the size of binary file is 14 bytes.

If assembler find error during 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 for writing assembly instructions.

3.1- Number syntax:

The assembler for now only support Hexadecimal numbers.
Hexadicimal must be in UPPERCASE and from 0x00 to 0xFFFFFFFF
or you get message : "error: invalid operand."
for example :
0xF3, 0x23FA, 0x0016, 0xFFFFFFFF

3.2- Instruction syntax:

Minis assembler accept only one instruction at each line followed
by line break or comment.

Name of Registers :
Until now Minis assembler support only this registers.

General Registers:
8-bit registers:
al, cl, dl, bl, ah, ch, dh, bh
16-bit registers:
ax, cx, dx, bx, sp, bp, si, di
32-bit registers:
eax, ecx, edx, ebx, esp, ebp, esi, edi

Segment Registers:
cs, ss, ds, es, fs, gs

Index Registers:
eax, ecx, edx, ebx, esi, edi, bx, si, di

Control Registers:
cr0, cr2, cr3

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.
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:

Jump instruction must followed by operand size (byte, word, dword)

Unconditional jump :
Unconditional relative jmp must be followed by operand size (byte, word, dword)
For example :
```
        jmp byte here
        nop
        here:
jmp 0x1000: word 0x0000  ; 16-bit far jump
jmp 0x8: dword 0x10050    ; 32-bit far jump
jmp word bx       ; jmp at address in bx register
jmp dword eax   ; jmp at address in eax register
```

Conditional jump :
Conditional relative jmp must be followed by operand size (byte)
For example :
```
        je byte there
        nop
        there:
```

3.8- Calls:

The assembler for now only support operand size (word, dword)
and call instruction must followed by operand size.

For example :
```
call word exe_this ; run code at label (exe_this)
; and return here after call instruction
exe_this:
mov si, 10
ret
call word bx  ; call address in bx register
call dword eax  ; call address in eax register
```

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 until now support only transfer between registers or from immediate value to register, or from immediate value to memory using index registers and imposible from memory to memory.

The instruction 'mov reg8, byte [index]' move data at index register to 8-bit general registers
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 at address ds:si to dl register.
mov ax, word [es:bx] ; this copy two bytes at address es:bx to ax register.

reg8/reg16/reg32 : is 8/16/32-bit general registers
imm8/imm16/imm32 : is immediate byte/word/dword or label
index : is one of the index registers
seg : is one of the segment register
cr : is one of the control registers

To copy data to or from a register use the following rules:
mov reg8, reg8
mov reg8, imm8
mov reg8, byte [index]
mov byte [index], reg8
mov byte [index], imm8 (hexadecimal)

mov reg16, reg16
mov reg16, imm16
mov reg16, word [index]
mov word [index], reg16
mov word [index], imm16 (hexadecimal)

mov reg32, reg32
mov reg32, imm32
mov reg32, dword [index]
mov dword [index], reg32
mov dword [index], imm32 (hexadecimal)

```
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
```

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
```

4- Instruction supported:

aaa
aad
aam
aas
adc
add
and
call
cbw
cwde
clc

cld
cli
clts
cmc
cmp
cmpsb
cmpsw
cmpsd
cwd
cdq
daa
das
dec
div
hlt
idiv
imul
in
inc
insb
insw
insd
int
int3
into
iret
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
leave
lgdt
lidt
lock
lodsb
lodsw
lodsd
loop
mov
movsb
movsw
movsd
movsx
movzx
mul
neg
nop
not
or
out
outsb
outsw
outsd
pop
popa
popad
popf
push

pusha
pushad
pushf
rep
repe
repz
repne
repnz
ret
sahf
shl
shr
sbb
scasb
scasw
scasd
stc
std
sti
stosb
stosw
stosd
sub
test
wait
xlatb
xor