免費電子書

學習

Intel x86 Assembly Language & Microarchitecture

Free unaffiliated eBook created from
Stack Overflow contributors.

#x86

1
1: x86 2
2
Examples 2
x86 2
x86 Linux Hello World 3
2: 5
5
5
Examples 5
5
Carry 5
5
'sbb' 5
5
5
0 6
6
test 6
6
6
Linux 6
35 7
7
lea 7
7
7
3: - 8
Examples 8
8
8

8
8
8
8
8
8
8
8
8
8
8
8
8
8
8
8
9
9
80386 9
9
9
PDBR PDBR 10
10
80486 10
10
10
10
PAE 11
11
11
11
PSE 11

PSE-32PSE-40 11
4: **13**
13
13
Examples 13
32cdecl 13
13
13
Clobbered **13**
64V 13
13
14
Clobbered **14**
32stdcall 14
14
14
Clobbered **14**
32cdecl - 14
8,16,32 **14**
64 **14**
15
32cdecl - 16
floatdouble **16**
16
17
64Windows 17
17
17
Clobbered **17**
18

32cdecl - 18
18
18
18
18
5: 20
20
20
Examples 22
22
6: 29
29
Examples 29
IA-32GAScdecl 29
MS-DOSTASM / MASM16 30
16 30
30
31
31
NASM 33
MS-DOSTASM / MASM16 33
33
33
33
33
35
NASM 35
35
MS-DOSTASM / MASM16 36
16 36
36

36

36

NASM 37

7: 38

Examples 38

Microsoft Assembler - MASM 38

38

ATT - 38

BorlandTurbo Assembler - TASM 39

GNU - 39

Netwide Assembler - NASM 39

- YASM 40

8: 41

Examples 41

41

41

41

41

41

41

42

42

42

43

43

43

43

43

44

44

45

45

45
45
a_label 46
46
46
9: 47
47
47
Examples 47
MOV 47
10: 49
Examples 49
49
49
49
49
49
/ 49
50
50
50
50
50
51
11: 54
Examples 54
BIOS 54
BIOS 54
BIOS 54
54
54

54

CHS 54

RTC 55

RTC 55

RTC 55

55

56

56

56

12: **57**

Examples 57

16 57

57

32 57

8 58

58

58

58

58

58

64 59

59

59

FLAGS **60**

60

80286 61

80386 61

80486 61

61

62

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: intel-x86-assembly-language---microarchitecture

It is an unofficial and free Intel x86 Assembly Language & Microarchitecture ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Intel x86 Assembly Language & Microarchitecture.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1: x86

x86。

x86。 x86Documentation。

Examples

x86

x86Intel 8086。 208016。

x86。 C。 “”。 。

NASMNetwide Assemblerhttp://nasm.us/。 NASM。

3264NASMWindows。 NASMWindows。

Linux

NASMLinux。

```
nasm -v
```

。 NASMNASM。 DebianUbuntu

```
sudo apt-get install nasm
```

RPM

```
sudo yum install nasm
```

Mac OS X.

OS XYosemiteEl CapitanNASM。 El Capitan0.98.40。 。 NASM2.112.12。

NASMOS X。

NASM。 / usr / local

```
$ sudo su
<user's password entered to become root>
# cd /usr/local/bin
# cp <path/to/unzipped/nasm/files/nasm> ./
# exit
```

NASM`/usr/local/bin` 。

```
$ echo 'export PATH=/usr/local/bin:$PATH' >> ~/.bash_profile
```

/usr/local/bin。 nasm -v。

x86 Linux Hello World

32x86 LinuxNASMHello Worldlibc。 。 ; 。

UnixasmCC ++。 POSIXAPIABI。

---

write(2)_exit(2) exit(3) libcstdio。 _exit()sys_exit_groupsys_exit 。 syscalls(2)libclibc。

argseax int 0x80。 AssemblyCasm。

32ABI/usr/include/i386-linux-gnu/asm/unistd_32.h /usr/include/x86_64-linux-gnu/asm/unistd_32.h 。

#include <sys/syscall.h>echo '#include <sys/syscall.h>' | gcc -E - -dM | lessdefs Casm

---

```
section .text             ; Executable code goes in the .text section
global _start             ; The linker looks for this symbol to set the process entry point,
so execution start here
;;;a name followed by a colon defines a symbol.  The global _start directive modifies it so
it's a global symbol, not just one that we can CALL or JMP to from inside the asm.
;;; note that _start isn't really a "function".  You can't return from it, and the kernel
passes argc, argv, and env differently than main() would expect.
 _start:
    ;;; write(1, msg, len);
    ; Start by moving the arguments into registers, where the kernel will look for them
    mov     edx,len       ; 3rd arg goes in edx: buffer length
    mov     ecx,msg       ; 2nd arg goes in ecx: pointer to the buffer
    ;Set output to stdout (goes to your terminal, or wherever you redirect or pipe)
    mov     ebx,1         ; 1st arg goes in ebx: Unix file descriptor. 1 = stdout, which is
normally connected to the terminal.

    mov     eax,4         ; system call number (from SYS_write / __NR_write from unistd_32.h).
    int     0x80          ; generate an interrupt, activating the kernel's system-call
handling code.  64-bit code uses a different instruction, different registers, and different
call numbers.
    ;; eax = return value, all other registers unchanged.

    ;;;Second, exit the process.  There's nothing to return to, so we can't use a ret
instruction (like we could if this was main() or any function with a caller)
    ;;; If we don't exit, execution continues into whatever bytes are next in the memory page,
    ;;; typically leading to a segmentation fault because the padding 00 00 decodes to  add
[eax],al.

    ;;; _exit(0);
    xor     ebx,ebx       ; first arg = exit status = 0.  (will be truncated to 8 bits).
Zeroing registers is a special case on x86, and mov ebx,0 would be less efficient.
                      ;; leaving out the zeroing of ebx would mean we exit(1), i.e. with an
error status, since ebx still holds 1 from earlier.
    mov     eax,1         ; put __NR_exit into eax
    int     0x80          ;Execute the Linux function
```

```
section     .rodata       ; Section for read-only constants

            ;; msg is a label, and in this context doesn't need to be msg:.  It could be on a
separate line.
            ;; db = Data Bytes: assemble some literal bytes into the output file.
msg     db  'Hello, world!',0xa     ; ASCII string constant plus a newline (0x10)

            ;;  No terminating zero byte is needed, because we're using write(), which takes
a buffer + length instead of an implicit-length string.
            ;; To make this a C string that we could pass to puts or strlen, we'd need a
terminating 0 byte. (e.g. "...", 0x10, 0)

len     equ $ - msg       ; Define an assemble-time constant (not stored by itself in the
output file, but will appear as an immediate operand in insns that use it)
                          ; Calculate len = string length.  subtract the address of the start
                          ; of the string from the current position ($)
  ;; equivalently, we could have put a str_end: label after the string and done   len equ
str_end - str
```

Linux`Hello.asm`32

```
nasm -felf32 Hello.asm                  # assemble as 32-bit code.  Add -Worphan-labels -g -
Fdwarf  for debug symbols and warnings
gcc -nostdlib -m32 Hello.o -o Hello     # link without CRT startup code or libc, making a
static binary
```

3264LinuxNASM / YASMGNU ATTGNU `as` ◦ 6432`-m32`◦

`strace`

```
$ strace ./Hello
execve("./Hello", ["./Hello"], [/* 72 vars */]) = 0
[ Process PID=4019 runs in 32 bit mode. ]
write(1, "Hello, world!\n", 14Hello, world!
)         = 14
_exit(0)                                = ?
+++ exited with 0 +++
```

stderrstdout`write`◦ ◦ syscallgdb◦

x86-64args◦ `syscallint 0x80` ◦

x86 https://riptutorial.com/zh-TW/x86/topic/1164/x86

2:

x86 - ◦ - ◦ ◦ ◦

64IA-32 x86◦

Examples

MOV0

```
B8 00 00 00 00    MOV eax, 0
```

5◦

MOVXOR

```
33 C0             XOR eax, eax
```

2 ◦

Carry

Carry C

```
    mov  al, 1
    jc   NotZero
    mov  al, 0
NotZero:
```

'sbb'

“”

```
    sbb  al,al    ; Move Carry to al
```

Cal◦ 0xFF -1 ◦ 0x01

```
    and  al, 0x01 ; Mask down to 1 or 0
```

-
-
-

-
-

0

```
    cmp   eax, 0
```

```
83 F8 00      cmp   eax, 0
```

test

```
    test   eax, eax      ; Equal to zero?
```

```
85 c0         test   eax, eax
```

-
-

。

Linux

32Linuxsysenter`int 0x80`。
32Linux

```
mov eax, <System call number>
mov ebx, <Argument 1> ;If applicable
mov ecx, <Argument 2> ;If applicable
mov edx, <Argument 3> ;If applicable
push <label to jump to after the syscall>
push ecx
push edx
push ebp
mov ebp, esp
sysenter
```

。
ebpesp33212。ebpedxecxLEAESP。

```
mov eax, <System call number>
mov ebx, <Argument 1>
mov ecx, <Argument 2>
mov edx, <Argument 3>
push <label to jump to after the syscall>
lea ebp, [esp-12]
sysenter
```

sys_exit

```
mov eax, 1
```

```
xor ebx, ebx ;Set the exit status to 0
mov ebp, esp
sysenter
```

35

```
    imul ecx, 3      ; Set ecx to 5 times its previous value
    imul edx, eax, 5 ; Store 5 times the contend of eax in edx
```

lea

◦ ◦ 3264esprsp 35lea◦ ◦

```
    lea ecx, [2*ecx+ecx] ; Load 2*ecx+ecx = 3*ecx into ecx
    lea edx, [4*edx+edx] ; Load 4*edx+edx = 5*edx into edx
```

```
    lea ecx, [3*ecx]
    lea edx, [5*edx]
```

ebprbp imul◦

-
- ebprbpimul
-
-

https://riptutorial.com/zh-TW/x86/topic/3215/

3: -

Examples

CPU。 。

/ - “”。 - - 。

1000。 “” - 。 。

。 -

。 - 。

Segmentation。 Segment。

- 。 Segments - 。

“”。

。 “”IntelARMMIPSPower。

。 。 。

- “”。 “”“”。

“” - 。 “”。

- 。 - 。

4 GB32。 。 1000“”。 。

“”。 “” - 。

- 。 - 。
- 。 “”。
- 。 。
- 。

- MessageBox。 Mac -

PagingCPUPaging。

PagingAddress。

```
+-----------------+------------+
| Page index      | Byte index |
+-----------------+------------+
```

。

。“”

```
+-----------+------------+------------+
| Dir index | Page index | Byte index |
+-----------+------------+------------+
```

Directory“”。

CPU - 。;;。CPU。

“” - 。Pages。

Pages。 - 。 - 。 。

TLBOS。TLB - 。

80386

803863232。 Paging4K32--101012

```
+-----------+------------+------------+
| Dir index | Page index | Byte index |
+-----------+------------+------------+
 3         2 2          1 1          0  Bit
 1         2 1          2 1          0  number
```

124K。101,024 - 44K

- 1,024 - 。
- 1,024 - 。
- 。

1,024。 。 32 - Page20。12;;

```
+--------------+----+------+-----+---+---+
| Page Address | OS | Used | Sup | W | P |
```

```
+--------------+----+------+-----+---+---+
Page Address = Top 20 bits of Page Table or Page address
OS           = Available for OS use
Used         = Whether this page has been accessed or written to
Sup          = Whether this page is Supervisory - only accessible by the OS
W            = Whether this page is allowed to be Written
P            = Whether this page is even Present
```

P0 -

PDBR PDBR

CPU。 CR3 。 PDBR 。

CPU。

1. 10PDBR;
2. 10;
3. 12。

1.2.

- “”;
- “” - ;
- “” - 。

14。 ;;。

。 。 。

80486

80486 Paging Subsystem80386。 - 。

“80386”。

Pentium。 。

Pentium - 4 MB4 MB1,0244K。

```
+-----------+----------------------+
| Dir Index | 4MB Byte Index       |
+-----------+----------------------+
 3         2 2                    0   Bit
 1         2 1                    0   number
```

```
+-----------+----+---+------+-----+---+---+
```

```
| Page Addr | OS | S | Used | Sup | W | P |
+-----------+----+---+------+-----+---+---+
Page Addr = Top 20 bits of Page Table or Page address
OS        = Available for OS use
S         = Size of Next Level: 0 = Page Table, 1 = 4 MB Page
Used      = Whether this page has been accessed or written to
Sup       = Whether this page is Supervisory - onlly accessible by the OS
W         = Whether this page is allowed to be Written
P         = Whether this page is even Present
```

- 4 MB4 MB4K4K。
- 4 MB - 。

。

PAE

PC。“” - “”“” - “”。

RAM - 32。Pentium ProPentium M64。 - 。

32。3264。4K6452122012。

644K5121,024。32

```
+-----+-----------+------------+------------+
| DPI | Dir Index | Page Index | Byte Index |
+-----+-----------+------------+------------+
 3   3 2         2 2          1 1          0   Bit
 1   0 9         1 0          2 1          0   number

 DPI        = 2-bit index into Directory Pointer Table
 Dir Index  = 9-bit index into Directory
 Page Index = 9-bit index into Page Table
 Byte Index = 12-bit index into Page (as before)
```

PDPT64。 PDBR CR3 PDPT - CR3324 GBRAM。 CR3PDPT32。

PSE

4MB。10 + 124MB9 + 122MB。

PSE-32PSE-40

Pentium ProPentum MPAEIntelPentium II“”32。

4MB

```
+-----------+------------+---------+
| Dir Index |  Unused    | Control |
+-----------+------------+---------+
```

DirPage Index。*31*

```
+-----------+------+-----+---------+
| Dir Index |Unused|Upper| Control |
+-----------+------+-----+---------+
```

PAE4 GBRAM - RAM4GB。43636PSE-36。。

4GB4GB - 4K。 - PAELinux。

AMDPSE8“PSE-40”

- https://riptutorial.com/zh-TW/x86/topic/3218/---

4:

/ Agner Fog 。 x86 ABIs x86-64 WindowsSystem VLinux。

- SystemV x86-64 ABI 。 Windows。 github wikiHJ Lu3264x32。 ABI/。 clang / gcc sign / zero narrow args32 ABI。 Clang。
- SystemV 32biti386ABI LinuxUnix。 。
- OS X 32x86 。 64System V. AppleFreeBSD pdf。
- Windows x86-64 `__fastcall`
- Windows `__vectorcall` 3264
- Windows 32bit `__stdcall` Win32 API。 `__cdecl` 。
- Windows64x86-64 SysV ABIAMD。

Examples

32cdecl

cdeclWindows 32POSIXi386 System V ABI 。 。

。 。

EAXEDXEAX64。st0x87。 。 EAX。

Clobbered

EBXEDIESIEBPESPFP / SSE。

EAXECXEDXFLAGSDFx87;。

64V.

POSIX64。

8RDIRSIRDXRCXR8R9R10R11。。。

RAX。。

Clobbered

RBPRBXR12-R15。。

32stdcall

stdcall32Windows API。

。。

EAX。

Clobbered

EMAECXEDX。EBXESIEDIEBP。

32cdecl -

8,16,32

8,16,3232[1] 。

。

。

```
//C prototype of the callee
void  __attribute__((cdecl)) foo(char a, short b, int c, long d);

foo(-1, 2, -3, 4);


;Call to foo in assembly

push DWORD 4              ;d, long is 32 bits, nothing special here
push DWORD 0ffffffdh      ;c, int is 32 bits, nothing special here
push DWORD 0badb0002h     ;b, short is 16 bits, higher WORD can be any value
push DWORD 0badbadffh     ;a, char is 8 bits, higher three bytes can be any value
call foo
add esp, 10h              ;Clean up the stack
```

64

64littel endian[2] 32。

```
//C prototype of the callee
void  __attribute__((cdecl)) foo(char a, short b, int c, long d);

foo(0x0123456789abcdefLL);

;Call to foo in assembly

push DWORD 89abcdefh        ;Higher DWORD of 0123456789abcdef
push DWORD 01234567h        ;Lower DWORD of 0123456789abcdef
call foo
add esp, 08h
```

AL8eax。

AX16eax。

EAX32。

EDX:EAX64EAX32EDX。

```
//C
char foo() { return -1; }

;Assembly
mov al, 0ffh
ret

//C
unsigned short foo() { return 2; }

;Assembly
mov ax, 2
ret

//C
int foo() { return -3; }

;Assembly
mov eax, 0fffffffdh
ret

//C
int foo() { return 4; }

;Assembly
xor edx, edx            ;EDX = 0
mov eax, 4            ;EAX = 4
ret
```

---

[1]4。x86 CPU24。

[2]DWORD

32cdecl -

floatdouble

32。
64Little Endian[1] 32。

```
//C prototype of callee
double foo(double a, float b);

foo(3.1457, 0.241);

;Assembly call

;3.1457 is 0x40092A64C2F837B5ULL
;0.241 is 0x3e76c8b4

push DWORD  3e76c8b4h        ;b, is 32 bits, nothing special here
push DWORD 0c2f837b5h        ;a, is 64 bits, Higher part of 3.1457
push DWORD  40092a64h        ;a, is 64 bits, Lower part of 3.1457
call foo
add esp, 0ch

;Call, using the FPU
;ST(0) = a, ST(1) = b
sub esp, 0ch
fstp QWORD PTR [esp]        ;Storing a as a QWORD on the stack
fstp DWORD PTR [esp+08h]    ;Storing b as a DWORD on the stack
call foo
add esp, 0ch
```

80[2]TBYTE32164 + 4 + 2 = 1041232。
Little Endian79-64[3] 63-3231-0。

```
//C prototype of the callee
void  __attribute__((cdecl)) foo(long double a);

foo(3.1457);

;Call to foo in assembly
;3.1457 is 0x4000c9532617c1bda800

push DWORD 4000h        ;Bits 79-64, as 32 bits push
push DWORD 0c9532617h        ;Bits 63-32
push DWORD 0c1bda800h        ;Bits 31-0
call foo
add esp, 0ch

;Call to foo, using the FPU
;ST(0) = a
```

```
sub esp, 0ch
fstp TBYTE PTR [esp]        ;Store a as ten byte on the stack
call foo
add esp, 0ch
```

`ST(0)` [4]。

```
//C
float one() { return 1; }

;Assembly
fld1            ;ST(0) = 1
ret

//C
double zero() { return 0; }

;Assembly
fldz            ;ST(0) = 0
ret

//C
long double pi() { return PI; }

;Assembly
fldpi            ;ST(0) = PI
ret
```

---

[1]DWORD。

[2]TBYTETen Bytes。

[3]WORD。

[4] TBYEFP。

64Windows

4RCXRDXR8R9。 XMM0XMM3。

。

64。

44QWORD。 。

RAX。64RAX。

Clobbered

RCXRDXR8R9XMM0XMM3RAXR10R11XMM4XMM5。。

16。“”816n + 816。

。

5amd64 Raymond Chen

32cdecl -

```
struct t
{
    int a, b, c, d;     // a is at offset 0, b at 4, c at 8, d at 0ch
    char e;             // e is at 10h
    short f;            // f is at 12h (naturally aligned)
    long g;             // g is at 14h
    char h;             // h is at 18h
    long i;             // i is at 1ch (naturally aligned)
};
```

。32;32cdecl。

。

```
int __attribute__((cdecl)) foo(struct t a);

struct t s = {0, -1, 2, -3, -4, 5, -6, 7, -8};
foo(s);
```

```
; Assembly call

push DWORD 0ffffffh8h    ; i (-8)
push DWORD 0badbad07h    ; h (7), pushed as DWORD to naturally align i, upper bytes can be
garbage
push DWORD 0ffffffah    ; g (-6)
push WORD 5              ; f (5)
push WORD 033fch         ; e (-4), pushed as WORD to naturally align f, upper byte can be
garbage
push DWORD 0ffffffdh    ; d (-3)
push DWORD 2             ; c (2)
push DWORD 0ffffffffh    ; b (-1)
push DWORD 0             ; a (0)
call foo
add esp, 20h
```

1 。 `struct S *retval` `struct S` `struct S`。

`eax`;`eax` `call`。

```
struct S
{
    unsigned char a, b, c;
};

struct S foo();         // compiled as struct S* foo(struct S* _out)
```

。

```
sub esp, 04h        ; allocate space for the struct

; call to foo
push esp            ; pointer to the output buffer
call foo
add esp, 00h        ; still as no parameters have been passed
```

。

```
struct S foo()
{
    struct S s;
    s.a = 1; s.b = -2; s.c = 3;
    return s;
}
```

```
; Assembly code
push ebx
mov eax, DWORD PTR [esp+08h]   ; access hidden parameter, it is a pointer to a buffer
mov ebx, 03fe01h               ; struct value, can be held in a register
mov DWORD [eax], ebx           ; copy the structure into the output buffer
pop ebx
ret 04h                        ; remove the hidden parameter from the stack
                               ; EAX = pointer to the output buffer
```

---

1 “”32。 `eax`。 LinuxGCC

WindowscdeclSystem V ABI“”32。 `eax`、`edx`64。 MSVCClangWin32。

https://riptutorial.com/zh-TW/x86/topic/3261/

5:

LAPIC	*APIC BASE*
APIC ID	+ 20H
	+ 0F0H
ICR;0-31	+ 300H
ICR;32-63	+ 310H

LAPIC*APIC BaseIA32_APIC_BASE* 。

。

LAPIC/“”。

。

- 1
- 。
-
- 。
- NASM ORG。 *ORG* 。

CPU *LAPIC* 。
APICLAPICeIOAPICxAPIC。

- 810。

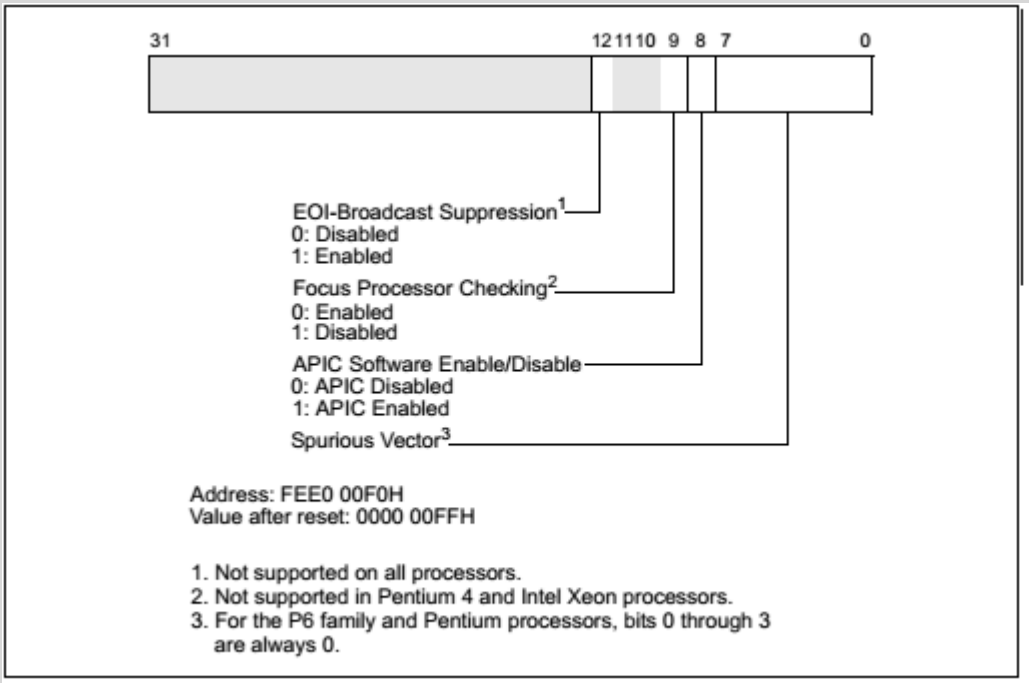


Figure 10-23. Spurious-Interrupt Vector Register (SVR)

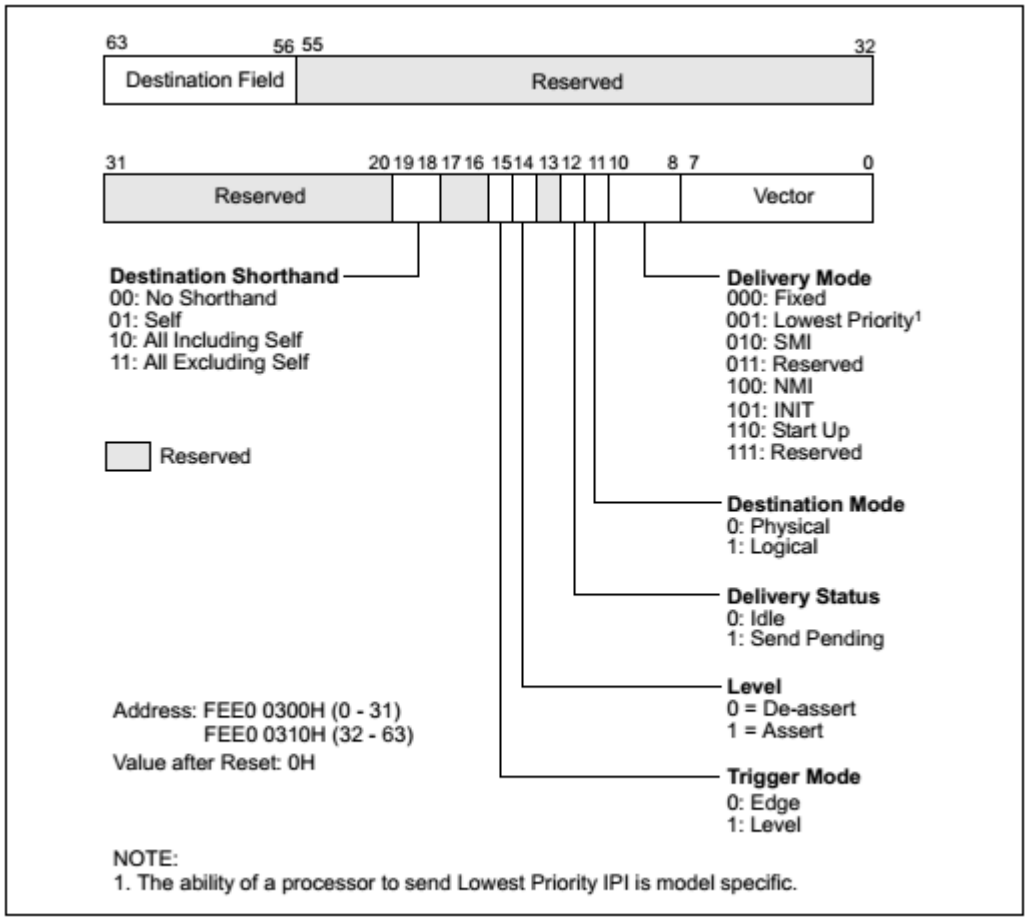


Figure 10-12. Interrupt Command Register (ICR)

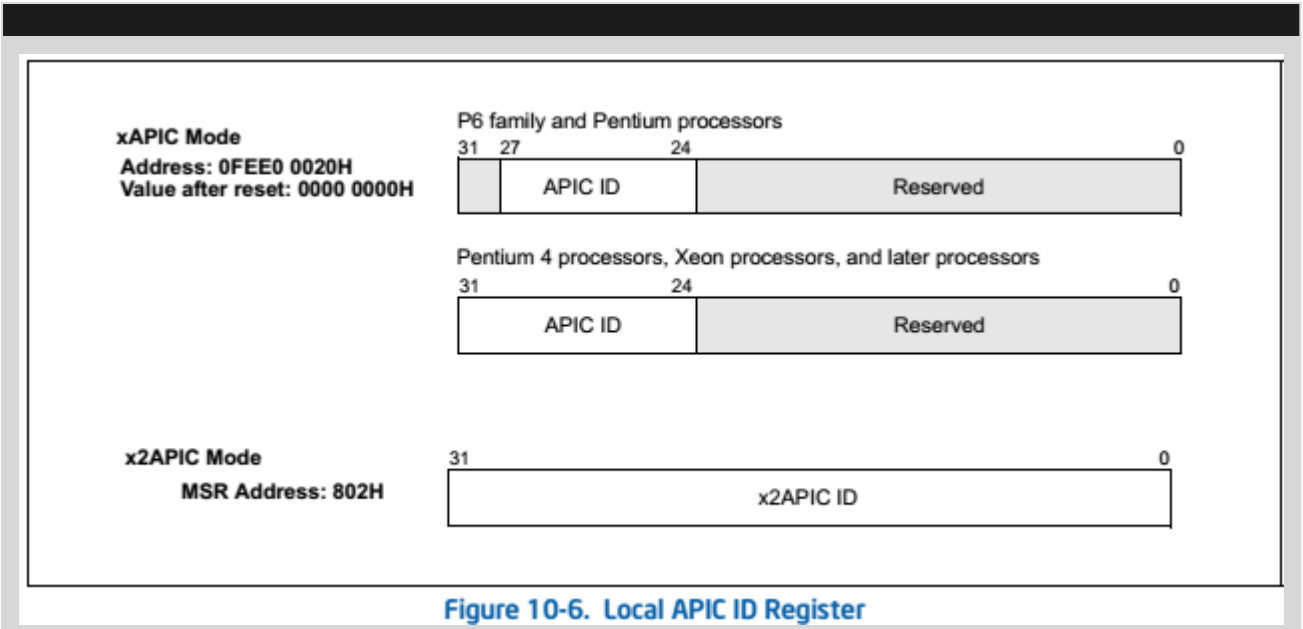


Figure 10-6. Local APIC ID Register

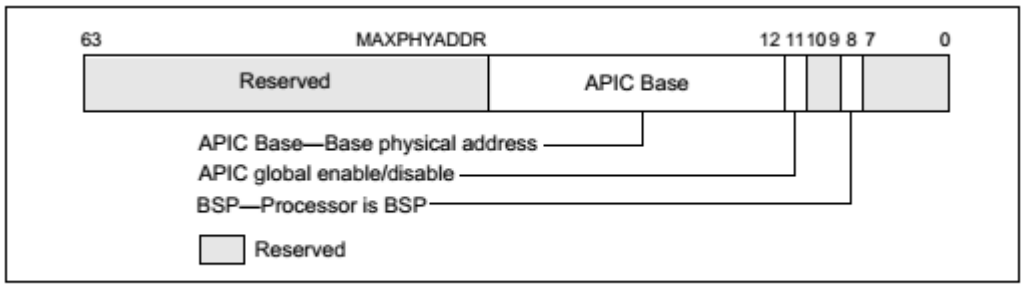


Figure 10-5. IA32_APIC_BASE MSR (APIC_BASE_MSR in P6 Family)

MSR	
IA32_APIC_BASE	1BH

1 。

Examples

AP BSPLAPIC ID。

```
; Assemble boot sector and insert it into a 1.44MiB floppy image
;
; nasm -f bin boot.asm -o boot.bin
; dd if=/dev/zero of=disk.img bs=512 count=2880
; dd if=boot.bin of=disk.img bs=512 conv=notrunc

BITS 16
; Bootloader starts at segment:offset 07c0h:0000h
section bootloader, vstart=0000h
jmp 7c0h:__START__

__START__:
 mov ax, cs
 mov ds, ax
 mov es, ax
 mov ss, ax
```

```
xor sp, sp
cld

;Clear screen
mov ax, 03h
int 10h

;Set limit of 4GiB and base 0 for FS and GS
call 7c0h:unrealmode

;Enable the APIC
call enable_lapic

;Move the payload to the expected address
mov si, payload_start_abs
mov cx, payload_end-payload + 1
mov di, 400h                ;7c0h:400h = 8000h
rep movsb


;Wakeup the other APs

;INIT
call lapic_send_init
mov cx, WAIT_10_ms
call us_wait

;SIPI
call lapic_send_sipi
mov cx, WAIT_200_us
call us_wait

;SIPI
call lapic_send_sipi

;Jump to the payload
jmp 0000h:8000h

;Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll
;  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll
;Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll

;CX = Wait (in ms) Max 65536 us (=0 on input)
us_wait:
 mov dx, 80h              ;POST Diagnose port, 1us per IO
 xor si, si
 rep outsb

 ret

 WAIT_10_ms     EQU 10000
 WAIT_200_us    EQU 200

;Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll
;  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll
;Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll


enable_lapic:

 ;Enable the APIC globally
```

```
 ;On P6 CPU once this flag is set to 0, it cannot be set back to 16
 ;Without an HARD RESET
 mov ecx, IA32_APIC_BASE_MSR
 rdmsr
 or ah, 08h        ;bit11: APIC GLOBAL Enable/Disable
 wrmsr

 ;Mask off lower 12 bits to get the APIC base address
 and ah, 0f0h
 mov DWORD [APIC_BASE], eax

 ;Newer processors enables the APIC through the Spurious Interrupt Vector register
 mov ecx, DWORD [fs: eax + APIC_REG_SIV]
 or ch, 01h                                ;bit8: APIC SOFTWARE enable/disable
 mov DWORD [fs: eax+APIC_REG_SIV], ecx

 ret

;Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll
;  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll
;Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll

lapic_send_sipi:
 mov eax, DWORD [APIC_BASE]

 ;Destination field is set to 0 has we will use a shorthand
 xor ebx, ebx
 mov DWORD [fs: eax+APIC_REG_ICR_HIGH], ebx

 ;Vector: 08h (Will make the CPU execute instruction ad address 08000h)
 ;Delivery mode: Startup
 ;Destination mode: ignored (0)
 ;Level: ignored (1)
 ;Trigger mode: ignored (0)
 ;Shorthand: All excluding self (3)
 mov ebx, 0c4608h
 mov DWORD [fs: eax+APIC_REG_ICR_LOW], ebx  ;Writing the low DWORD sent the IPI

 ret

 ;Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll
;  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll
;Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll

lapic_send_init:
 mov eax, DWORD [APIC_BASE]

 ;Destination field is set to 0 has we will use a shorthand
 xor ebx, ebx
 mov DWORD [fs: eax+APIC_REG_ICR_HIGH], ebx

 ;Vector: 00h
 ;Delivery mode: Startup
 ;Destination mode: ignored (0)
 ;Level: ignored (1)
 ;Trigger mode: ignored (0)
 ;Shorthand: All excluding self (3)
 mov ebx, 0c4500h
 mov DWORD [fs: eax+APIC_REG_ICR_LOW], ebx  ;Writing the low DWORD sent the IPI

 ret
```

```
 IA32_APIC_BASE_MSR    EQU    1bh

 APIC_REG_SIV        EQU    0f0h

 APIC_REG_ICR_LOW    EQU 300h
 APIC_REG_ICR_HIGH    EQU 310h

 APIC_REG_ID        EQU 20h

 ;Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll
 ;  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll
 ;Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll

 APIC_BASE            dd     00h

 ;Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll
 ;  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll
 ;Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll

unrealmode:
 lgdt [cs:GDT]

 cli

 mov eax, cr0
 or ax, 01h
 mov cr0, eax

 mov bx, 08h
 mov fs, bx
 mov gs, bx

 and ax, 0fffeh
 mov cr0, eax

 sti

 ;IMPORTAT: This call is FAR!
 ;So it can be called from everywhere
 retf

 GDT:
    dw 0fh
    dd GDT + 7c00h
    dw 00h

    dd 0000ffffh
    dd 00cf9200h

 ;Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll
 ;  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll
 ;Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll  Ll

payload_start_abs:
; payload starts at segment:offset 0800h:0000h
section payload, vstart=0000h, align=1
 payload:

  ;IMPORTANT NOTE: Here we are in a "new" CPU every state we set before is no
  ;more present here (except for the BSP, but we handler every processor with
```

```
 ;the same code).
jmp 800h: __RESTART__

__RESTART__:
 mov ax, cs
 mov ds, ax
 xor sp, sp
 cld

 ;IMPORTANT: We can't use the stack yet. Every CPU is pointing to the same stack!

 ;Get an unique id
 mov ax, WORD [counter]
 .try:
   mov bx, ax
   inc bx
   lock cmpxchg WORD [counter], bx
  jnz .try

 mov cx, ax            ;Save this unique id

 ;Stack segment = CS + unique id * 1000
 shl ax, 12
 mov bx, cs
 add ax, bx
 mov ss, ax

 ;Text buffer
 push 0b800h
 pop es

 ;Set unreal mode again
 call 7c0h:unrealmode

 ;Use GS for old variables
 mov ax, 7c0h
 mov gs, ax

 ;Calculate text row
 mov ax, cx
 mov bx, 160d          ;80 * 2
 mul bx
 mov di, ax

 ;Get LAPIC id
 mov ebx, DWORD [gs:APIC_BASE]
 mov edx, DWORD [fs:ebx + APIC_REG_ID]
 shr edx, 24d
 call itoa8

 cli
 hlt

 ;DL = Number
 ;DI = ptr to text buffer
 itoa8:
   mov bx, dx
   shr bx, 0fh
   mov al, BYTE [bx +  digits]
   mov ah, 09h
   stosw
```

```
    mov bx, dx
    and bx, 0fh
    mov al, BYTE [bx +  digits]
    mov ah, 09h
    stosw

    ret

  digits db "0123456789abcdef"
  counter dw 0

 payload_end:



; Boot signature is at physical offset 01feh of
; the boot sector
section bootsig, start=01feh
 dw 0aa55h
```

1.AP
AP*INIT-SIPI-SIPI* ISS。

ISSBSP AP。

SIPICPUSIPISIPI。。

SIPI 。
8*V* 16*vv* CPU*0vv000h*。
0vv000h WA。
WA4KiB。

08h*V* WA*08000h* 400h。

AP。

2.AP
WA。*7c00h* 。

。
AP

。
CPU。 CPUAPIC ID。

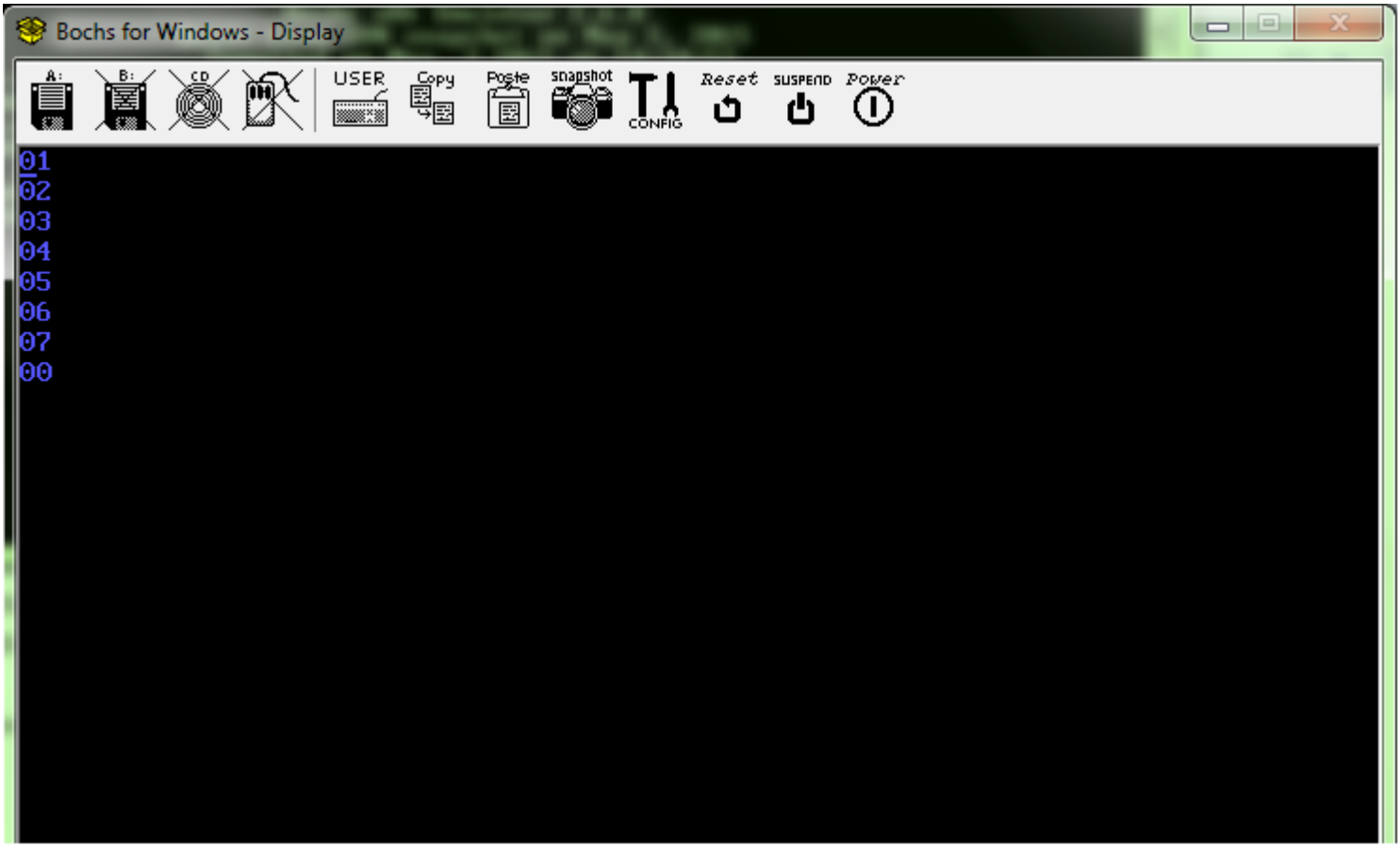
CPU*800h :(* 1000h*AP64KiB。
CPU 80 * 2 * 。

`lock cmpxchg`WORD。

- 80h1μs。

- `unrealmode`。
- BSP。

Bochs8



https://riptutorial.com/zh-TW/x86/topic/5809/

6:

。

。

Psuedo

```
function string_to_integer(str):
    result = 0
    for (each characters in str, left to right):
        result = result * 10
        add ((code of the character) - (code of character 0)) to result
    return result
```

0-9afAF。。

Examples

IA-32GAScdecl

```
# make this routine available outside this translation unit
.globl string_to_integer

string_to_integer:
    # function prologue
    push %ebp
    mov %esp, %ebp
    push %esi

    # initialize result (%eax) to zero
    xor %eax, %eax
    # fetch pointer to the string
    mov 8(%ebp), %esi

    # clear high bits of %ecx to be used in addition
    xor %ecx, %ecx
    # do the conversion
string_to_integer_loop:
    # fetch a character
    mov (%esi), %cl
    # exit loop when hit to NUL character
    test %cl, %cl
    jz string_to_integer_loop_end
    # multiply the result by 10
    mov $10, %edx
    mul %edx
    # convert the character to number and add it
    sub $'0', %cl
    add %ecx, %eax
    # proceed to next character
    inc %esi
    jmp string_to_integer_loop
string_to_integer_loop_end:
```

```
    # function epilogue
    pop %esi
    leave
    ret
```

GAS`%eax` `%esi`callee-save。

/。

C`unsigned int`4

```
#include <stdio.h>

unsigned int string_to_integer(const char* str);

int main(void) {
    const char* testcases[] = {
        "0",
        "1",
        "10",
        "12345",
        "1234567890",
        NULL
    };
    const char** data;
    for (data = testcases; *data != NULL; data++) {
        printf("string_to_integer(%s) = %u\n", *data, string_to_integer(*data));
    }
    return 0;
}
```

`string_to_integer_string_to_integer` C。

MS-DOSTASM / MASM16

16。

Int 21 / AH = 0Ah。

。

65535 = 2 16 - 1。

16。

AX。 ZF CF OF。

		ZF	CF	
16				
7FFFH				

ZF。

```
call read_uint16
jo _handle_overflow            ;Number too big (Optional, the test below will do)
jnz _handle_invalid            ;Number format is invalid

;Here AX is the number read
```

```
;Returns:
  ;
  ;If the number is correctly converted:
  ;   ZF = 1, CF = 0, OF = 0
  ;   AX = number
  ;
  ;If the user input an invalid digit:
  ;   ZF = 0, CF = 1, OF = 0
  ;   AX = Partially converted number
  ;
  ;If the user input a number too big
  ;   ZF = 0, CF = 1, OF = 1
  ;   AX = 07fffh
  ;
  ;ZF/CF can be used to discriminate valid vs invalid inputs
  ;OF can be used to discrimate the invalid inputs (overflow vs invalid digit)
  ;
  read_uint16:
    push bp
    mov bp, sp

    ;This code is an example in Stack Overflow Documentation project.
    ;x86/Converting Decimal strings to integers


    ;Create the buffer structure on the stack

    sub sp, 06h                ;Reserve 6 byte on the stack (5 + CR)
    push 0006h                 ;Header

    push ds
    push bx
    push cx
    push dx

    ;Set DS = SS

    mov ax, ss
    mov ds, ax


    ;Call Int 21/AH=0A

    lea dx, [bp-08h]            ;Address of the buffer structure
    mov ah, 0ah
    int 21h

    ;Start converting

    lea si, [bp-06h]
    xor ax, ax
```

```
 mov bx, 10
 xor cx, cx

_r_ui16_convert:

 ;Get current char

 mov cl, BYTE PTR [si]
 inc si

 ;Check if end of string

 cmp cl, CR_CHAR
 je _r_ui16_end                      ;ZF = 1, CF = 0, OF = 0

 ;Convert char into digit and check

 sub cl, '0'
 jb _r_ui16_carry_end                ;ZF = 0, CF = 1, OF = X -> 0
 cmp cl, 9
 ja _r_ui16_carry_end                ;ZF = 0, CF = 0 -> 1, OF = X -> 0


 ;Update the partial result (taking care of overflow)

 ;AX = AX * 10
 mul bx

 ;DX:AX = DX:AX + CX
 add ax, cx
 adc dx, 0

 test dx, dx
jz _r_ui16_convert            ;No overflow

 ;set OF and CF
 mov ax, 8000h
 dec ax
 stc

jmp _r_ui16_end                      ;ZF = 0, CF = 1, OF = 1

_r_ui16_carry_end:

 or bl, 1                ;Clear OF and ZF
 stc                     ;Set carry

 ;ZF = 0, CF = 1, OF = 0

_r_ui16_end:
 ;Don't mess with flags hereafter!

 pop dx
 pop cx
 pop bx
 pop ds

 mov sp, bp

 pop bp
 ret
```

```
    CR_CHAR EQU 0dh
```

NASM

NASM`PTR mov cl, BYTE PTR [si]mov cl, BYTE [si]`

MS-DOSTASM / MASM16

22 1 2 2 2 3 2 4 1。
2 LSb*n*。
16。 8。
16;53163.5 = 151[3]。

AND。
2。

a*D*b*S*。
。

。
16。
。

ñ	
	BASE2 BASE4 BASE8BASE16
。0“0”	

```
push 241
push BASE16
push 0
call print_pow2             ;Prints f1

push 241
push BASE16
push 1
call print_pow2             ;Prints 00f1

push 241
push BASE2
push 0
call print_pow2             ;Prints 11110001
```

TASM `EQU`*TASM*`/m` 。

```
;Parameters (in order of push):
;
;number
;base (Use constants below)
;print leading zeros
print_pow2:
 push bp
 mov bp, sp

 push ax
 push bx
 push cx
 push dx
 push si
 push di

 ;Get parameters into the registers

 ;SI = Number (left) to convert
 ;CH = Amount of bits to shift for each digit (D)
 ;CL = Amount od bits to shift the number (S)
 ;BX = Bit mask for a digit

 mov si, WORD PTR [bp+08h]
 mov cx, WORD PTR [bp+06h]            ;CL = D, CH = S

 ;Computes BX = (1 << D)-1

 mov bx, 1
 shl bx, cl
 dec bx

 xchg cl, ch              ;CL = S, CH = D

_pp2_convert:
 mov di, si
 shr di, cl
 and di, bx         ;DI = Current digit

 or WORD PTR [bp+04h], di             ;If digit is non zero, [bp+04h] will become non zero
                      ;If [bp+04h] was non zero, result is non zero
 jnz _pp2_print                       ;Simply put, if the result is non zero, we must print
the digit


 ;Here we have a non significant zero
 ;We should skip it BUT only if it is not the last digit (0 should be printed as "0" not
 ;an empty string)

 test cl, cl
 jnz _pp_continue


_pp2_print:
 ;Convert digit to digital and print it


 mov dl, BYTE PTR [DIGITS + di]
 mov ah, 02h
 int 21h
```

```
_pp_continue:
 ;Remove digit from the number

 sub cl, ch
jnc _pp2_convert

 pop di
 pop si
 pop dx
 pop cx
 pop bx
 pop ax

 pop bp
 ret 06h
```

```
This data must be put in the data segment, the one reached by `DS`.

DIGITS    db    "0123456789abcdef"

;Format for each WORD is S D where S and D are bytes (S the higher one)
;D = Bits per digit  --> log2(BASE)
;S = Initial shift count --> D*[ceil(16/D)-1]

BASE2    EQU    0f01h
BASE4    EQU    0e02h
BASE8    EQU    0f03h
BASE16    EQU    0c04h
```

NASM

NASMPTR `mov si, WORD PTR [bp+08h]mov si, WORD PTR [bp+08h]`

$2^{255}2^{16}16$。

1. `BASEx` $x2^{n}$。
   $DD = n$。
   S。$S = N \cdot \lceil 16/ \rceil - 1$。
2. `DIGITS`。

32

$D = 5S = 15$`BASE32 EQU 0f05h`。

`DIGITS db "0123456789abcdefghijklmnopqrstuv"`。

`DIGITS`。

---

[1]BB。$log_2 B$。$log_2 2^n = n$。

[2]$2\ 2^n$。

$^{3}B = 2^{n}n$ | 16 *n*16。 162*n*2。 *B*2 $^{2^{k}}log_{2}$ log_{2} *B* 。

MS-DOSTASM / MASM16

16

Int 21 / AH = 02h。
`div`101610 ^{4} 。

。
16。

16	
	0。 0“0”

```
push 241
push 0
call print_dec          ;prints 241

push 56
push 1
call print_dec          ;prints 00056

push 0
push 0
call print_dec          ;prints 0
```

```
;Parameters (in order of push):
;
;number
;Show leading zeros
print_dec:
 push bp
 mov bp, sp

 push ax
 push bx
 push cx
 push dx

 ;Set up registers:
 ;AX = Number left to print
 ;BX = Power of ten to extract the current digit
 ;DX = Scratch/Needed for DIV
 ;CX = Scratch

 mov ax, WORD PTR [bp+06h]
 mov bx, 10000d
 xor dx, dx

_pd_convert:
```

```
 div bx                          ;DX = Number without highmost digit, AX = Highmost digit
 mov cx, dx                      ;Number left to print

 ;If digit is non zero or param for leading zeros is non zero
 ;print the digit
 or WORD PTR [bp+04h], ax
 jnz _pd_print

 ;If both are zeros, make sure to show at least one digit so that 0 prints as "0"
 cmp bx, 1
 jne _pd_continue

_pd_print:

 ;Print digit in AL

 mov dl, al
 add dl, '0'
 mov ah, 02h
 int 21h

_pd_continue:
 ;BX = BX/10
 ;DX = 0

 mov ax, bx
 xor dx, dx
 mov bx, 10d
 div bx
 mov bx, ax

 ;Put what's left of the number in AX again and repeat...
 mov ax, cx

 ;...Until the divisor is zero
 test bx, bx
jnz _pd_convert

 pop dx
 pop cx
 pop bx
 pop ax

 pop bp
 ret 04h
```

NASM

NASM`PTRmov ax, WORD PTR [bp+06h]mov ax, WORD [bp+06h]`

https://riptutorial.com/zh-TW/x86/topic/3273/

7:

Examples

Microsoft Assembler - MASM

8086/8088IBM PCMASM。“”。

```
MaxSize      EQU     16          ; Define a constant
Symbol       DW      0x1234      ; Define a 16-bit WORD called Symbol to hold 0x1234

             MOV     AX, 10      ; AX now holds 10
             MOV     BX, MaxSize ; BX now holds 16
             MOV     CX, Symbol  ; ????
```

MOVSymbolCX SymbolCX CX0x12340x0102 CX0x1234 - OFFSET

```
             MOV     AX, [Symbol]      ; Contents of Symbol
             MOV     CX, OFFSET Symbol ; Address of Symbol
```

80868080,80084004。 。 。

。 dest source。 - 。 “” - 。

```
; Zero operand examples
NOP             ; No parameters
CBW             ; Convert byte in AL into word in AX
MOVSB           ; Move byte pointed to by DS:SI to byte pointed to by ES:DI
                ; SI and DI are incremented or decremented according to D bit

; Prefix examples
REP   MOVSB     ; Move number of bytes in CX from DS:SI to ES:DI
                ; SI and DI are incremented or decremented according to D bit

; One operand examples
NOT      AX     ; Replace AX with its one's complement
MUL      CX     ; Multiply AX by CX and put 32-bit result in DX:AX

; Two operand examples
MOV      AL, [0x1234] ; Copy the contents of memory location DS:0x1234 into AL register
```

。 “”LDM“”LDI 。 MOV - 。 .....

ATT -

8086IBM PCMicrosoftUnix。 ATTUnix。 - source dest。

ATTx86

- %
  %al

%bx
- $
  $4
- source dest
- movw $4, %ax ; Move word 4 into AX

BorlandTurbo Assembler - TASM

BorlandPascal“Turbo Pascal”。C / C ++PrologFortran。“Turbo Assembler”“TASM”。

TASMIDEALMASM。MASMMASM - BorlandMASM“”QUIRKS。

TASMMASMASM - IDEAL。

GNU -

GNUx86UnixATT/。

Netwide Assembler - NASM

NASMx86 - x86MacOS。

Intel“” - “”。

。

```
response:    db       'Y'     ; Character that user typed

             cmp      response, 'N' ; *** Error! Unknown size!
             cmp byte response, 'N' ; That's better!
             cmp      response, ax  ; No error!
```

NASM。 - “”“”。

NASM

```
       STRUC     Point
X      resw      1
Y      resw      1
       ENDSTRUC
```

XY。“”XY

```
       STRUC     Point
Pt_X   resw      1
Pt_Y   resw      1
       ENDSTRUC
```

NASM。“”

```
        STRUC       Point
.X      resw        1
.Y      resw        1
        ENDSTRUC

Cursor  ISTRUC      Point
        ENDISTRUC

        mov         ax,[Cursor+Point.X]
        mov         dx,[Cursor+Point.Y]
```

NASM

```
        mov         ax,[Cursor.X]
        mov         dx,[Cursor.Y]
```

- YASM

YASMNASMIntelATT。

https://riptutorial.com/zh-TW/x86/topic/2403/

8:

Examples

```
jmp a_label                     ;Jump to a_label
jmp bx                          ;Jump to address in BX
jmp WORD [aPointer]             ;Jump to address in aPointer
jmp 7c0h:0000h                  ;Jump to segment 7c0h and offset 0000h
jmp FAR WORD [aFarPointer]      ;Jump to segment:offset in aFarPointer
```

`jmp a_label`

- ◦ `CS` ◦
- *rel*[1] `IP = IP + rel` ◦

`EB <rel8>EB <rel16/32>` ◦

NASM `jmp SHORT a_label  jmp WORD a_labeljmp DWORD a_label`◦

`jmp bxjmp WORD [aPointer]`

- ◦ `CS` ◦
- *regmem* `IP = reg  IP = mem` ◦

`FF /4` ◦

`jmp 7c0h:0000h`

- ◦
- **absolute***offset* `CS = segment, IP = offset` ◦

`EA <imm32/48>`◦

NASM `jmp 7c0h: WORD 0000hjmp 7c0h: DWORD 0000h`◦

`jmp FAR WORD [aFarPointer]`

- **far**◦
- *mem* [2] `CS = mem[23:16/32], IP = [15/31:0]` ◦

`FF /5` ◦

NASM*16:16* `jmp FAR WORD [aFarPointer]`*16:32* `jmp FAR DWORD [aFarPointer]` ◦

- ◦

```
mov bx, target            ;BX = absolute address of target
jmp bx
```

- 。

1。

2*seg16off16seg16off32 16:1616:32* 。

。 。

x86EFLAGS。

subadd xorand “”。 *CF OF SF ZF AF PF*。 cmpxchg*ZF* 。

。

x86 。

。 sub eax, ebx

ZF	。 *EAX - EBX =0*⇒EAX= EBX	。 *EAX - EBX≠0*⇒EAX≠EBX
CF	MSb。 *EAX - EBX <0*⇒EAX<EBX	。 *EAX - EBX*≮0⇒EAX≮EBX
SF	MSb。	MSb。
	。	。
PF	。	。
AF	BCD。 4。	BCD。 4。

---

suband instructions。

cmptest。 。

sub	cmp
and	test

```
test eax, eax             ;and eax, eax
                          ;ZF = 1 iff EAX is zero

test eax, 03h             ;and eax, 03h
                          ;ZF = 1 if both bit[1:0] are clear
                          ;ZF = 0 if at least one of bit[1:0] is set

cmp eax, 241d             ;sub eax, 241d
                          ;ZF = 1 iff EAX is 241
                          ;CF = 1 iff EAX < 241
```

CPU[1]sign。 。 。

---

[1]。 。

CPU。 *Jcc* - [1] 。

。 jae jnbjnc*CF = 0* 。

。

cmp。 cmp。

ZF。 *ZF = 1* 。

```
je a_label           ;Jump if operands are equal
jz a_label           ;Jump if zero (Synonym)

jne a_label          ;Jump if operands are NOT equal
jnz a_label          ;Jump if not zero (Synonym)
```

je jz	ZF = 1
jne jnz	ZF = 0

...

CF = 0 。 *CF = 0ZF*。

```
jae a_label      ;Jump if above or equal (>=)
jnc a_label      ;Jump if not carry (Synonym)
jnb a_label      ;Jump if not below (Synonym)

ja a_label       ;Jump if above  (>)
jnbe a_label     ;Jump if not below and not equal (Synonym)
```

jae jnc jnb	CF = 0
ja jnbe	CF = 0ZF = 0

SF = 0 SF。 OF = 01SF = OF 。

ZFl。

```
jge a_label      ;Jump if greater or equal (>=)
jnl a_label      ;Jump if not less (Synonym)

jg a_label       ;Jump if greater (>)
jnle a_label     ;Jump if not less and not equal (Synonym)
```

`jge jnl`	SF = OF
`jg jnle`	SF = OFZF = 0

。

```
jbe a_label      ;Jump if below or equal (<=)
jna a_label      ;Jump if not above (Synonym)

jb a_label       ;Jump if below (<)
jc a_label       ;Jump if carry (Synonym)
jnae a_label     ;Jump if not above and not equal (Synonym)

;SIGNED

jle a_label      ;Jump if less or equal (<=)
jng a_label      ;Jump if not greater (Synonym)

jl a_label       ;Jump if less (<)
jnge a_label     ;Jump if not greater and not equal (Synonym)
```

`jbe jna`	CF = 1ZF = 1
`jb jc jnae`	CF = 1
`jle jng`	SF= OFZF = 1
`jl jnge`	SF= OF

`j<flag_name>`*flag_nameF CF* → *C PF* → *P* 。

`js`	SF = 1
`jns`	SF = 0
`jo`	OF = 1
`jno`	OF = 0

`jp` `jpe` e = even	PF = 1
`jnp` `jpo` o = odd	PF = 0

x86。`cx` `ecx`CPU1632。

`cx/ecx` `reploop`。

```
jcxz  a_label   ; jump if cx (16b mode) or ecx (32b mode) is zero
jecxz a_label   ; synonym of jcxz (recommended in source code for 32b target)
```

`jcxz` `jecxz`	cx = 016b
`jcxz` `jecxz`	ecx = 032b

1。

...

```
cmp eax, ebx
ja a_label
```

```
cmp eax, ebx
jae a_label
```

```
cmp eax, ebx
jb a_label
```

```
cmp eax, ebx
jbe a_label
```

```
cmp eax, ebx
je a_label
```

```
cmp eax, ebx
jne a_label
```

...

```
cmp eax, ebx
jg a_label
```

```
cmp eax, ebx
jge a_label
```

```
cmp eax, ebx
jl a_label
```

```
cmp eax, ebx
jle a_label
```

```
cmp eax, ebx
je a_label
```

```
cmp eax, ebx
jne a_label
```

---

a_label

“” `a_label`CPU。“”CPU。

。

`ja``jnbe`。

>	ja	jg
> =	jae	jge
<	jb	jl
<=	jbe	jle
=	je	je
≠=<>	jne	jne

https://riptutorial.com/zh-TW/x86/topic/5808/

9:

- ***.386*** **MASM**x86386。
- ***.model*** .MODEL 。
- ***.code*** 。
- ***proc*** 。
- ***ret*** 。
- ***endp*** 。
- ***public*** 。
- “ **end main** ”。
- ***call*** 。
- ***ecx*** 。
- ***ecx*** 。
- ***mul*** eax

mov。

Examples

MOV

`mov`。

/ [in] CPU。

source_of / destination_for。

`mov`。

3264x86 CPU。 `MOVSB`[s]。

1**MASM** Visual Studio 2015x86
2

```
.386
.model small
.code

public main
main proc
    mov ecx, 16       ; Move immediate value 16 into ecx
    mov eax, ecx      ; Copy value of ecx into eax
    ret               ; return back to caller
        ; function return value is in eax (16)
main endp
end main
```

3。

16 。

https://riptutorial.com/zh-TW/x86/topic/8030/

10:

Examples

x86808680881616。 16164 `CS DS ES SS` 。

`PUSHPOP` - 。

CPU

```
        +-------------------+---------+
Segment | 16-bit value      | 0 0 0 0 |
        +-------------------+---------+
                   PLUS
        +---------+-------------------+
Address | 0 0 0 0 | 16-bit value      |
        +---------+-------------------+
                  EQUALS
        +-----------------------------+
Result  |  20-bit memory address      |
        +-----------------------------+
```

- `CS DSSS`;
- `CS DSSS`4K - 1664K。

80286“”“”qv。

- 16。
- “” - 。

.....

802868086“”“”。 x863264。

“”。 Segment.....。 SegmentBaseLimitSegmentOS Kernel 。

16

```
+------------+-----+------+
| Desc Index | G/L | Priv |
+------------+-----+------+
 Desc Index = 13-bit index into a Descriptor Table (described below)
 G/L        = 1-bit flag for which Descriptor Table to Index: Global or Local
 Priv       = 2-bit field defining the Privilege level for access
```

/

/GDTLDT。 LDT - 。 。

64K8,1928。 8,192

-

- 。
- - 。 0x00000
  。 ;。
- 。 ;
  。 ;
  。 “” - 。
- 。 ;
  。 Segment - ;
  。 。

。 - ..... CPU。

13 - Windows.....

- ;
- ;
- ;
- ReadExecuted;
- 。

  Segment。
- GDT0x0000。 NULL ;
- 。
- 1,2,4。

。 CPU。

- 1. NULL;
  2. ;
  3. 。

。

- GDTR；

```
 GDT_Ptr    dw      SIZE GDT
            dd      OFFSET GDT

            ...

            lgdt    [GDT_Ptr]
```

- CR0PM

```
     mov   eax, cr0      ; Get CR0 into register
     or    eax, 0x01     ; Set the Protected Mode bit
     mov   cr0, eax      ; We're now in Protected Mode!
```

- GDT

```
     jmp   0x0008:NowInPM  ; This is a FAR Jump. 0x0008 is the Code Descriptor

NowInPM:
     mov   ax, 0x0010      ; This is the Data Descriptor
     mov   ds, ax
     mov   es, ax
     mov   ss, ax
     mov   sp, 0x0000      ; Top of stack!
```

CPU。。

- - A20；
- -。

。

AMD。

1. 。

。

2. 。。

GDT / LDT。

GDT04GiB。
32。

```
BITS 16

jmp 7c0h:__START__
```

```
__START__:
 push cs
 pop ds
 push ds
 pop ss
 xor sp, sp


 lgdt [GDT]            ;Set the GDTR register


 cli                ;We don't have an IDT set, we can't handle interrupts


 ;Entering protected mode

 mov eax, cr0
 or ax, 01h            ;Set bit PE (bit 0) of CR0
 mov cr0, eax            ;Apply

 ;We are now in Protected mode

 mov bx, 08h           ;Selector to use, RPL = 0, Table = 0 (GDT), Index = 1

 mov fs, bx            ;Load FS with descriptor 1 info
 mov gs, bx            ;Load GS with descriptor 1 info

 ;Exit protected mode

 and ax, 0fffeh            ;Clear bit PE (bit0) of CR0
 mov cr0, eax                  ;Apply

 sti

 ;Back to real mode

 ;Do nothing
 cli
 hlt



 GDT:
    ;First entry, number 0
    ;Null descriptor
    ;Used to store a m16&32 object that tells the GDT start and size

    dw 0fh                 ;Size in byte -1 of the GDT (2 descriptors = 16 bytes)
    dd GDT + 7c00h         ;Linear address of GDT start (24 bits)
    dw 00h                 ;Pad

    dd 0000ffffh           ;Base[15:00] = 0, Limit[15:00] = 0ffffh
    dd 00cf9200h           ;Base[31:24] = 0, G = 1, B = 1, Limit[19:16] = 0fh,
               ;P = 1, DPL = 0, E = 0, W = 1, A = 0, Base[23:16] = 00h


 TIMES 510-($-$$) db 00h
 dw 0aa55h
```

- ◦ `fsgs`“”16//◦

- `lgdt`GDT2432 ◦ ◦ `GDT+7c00h`◦
- MBRBPB `cs` / `ds` / `ss` TP *7C00h* 0*X7C00hX7c00h + X.*
- IDT◦
- hack6◦ `lgdt`... GDT◦

GDT“ ”3A3.4.3◦

https://riptutorial.com/zh-TW/x86/topic/3679/

11:

Examples

BIOS

BIOS

/BIOS。 BIOS 。

```
int <interrupt> ; interrupt must be a literal number, not in a register or memory
```

02550x00 - 0xFF。

BIOSAH“”AL。 AH。 BIOSAX16。 。

BIOS。

BIOS

BIOS

```
mov ah, <function>
mov al, <data>
int <interrupt>
```

```
mov ah, 0x0E            ; Select 'Write character' function
mov al, <char>          ; Character to write
int 0x10                ; Video services interrupt
```

```
mov ah, 0x00            ; Select 'Blocking read character' function
int 0x16                ; Keyboard services interrupt
mov <ascii_char>, al    ; AL contains the character read
mov <scan_code>, ah     ; AH contains the BIOS scan code
```

CHS

```
mov ah, 0x02            ; Select 'Drive read' function
mov bx, <destination>   ; Destination to write to, in ES:BX
mov al, <num_sectors>   ; Number of sectors to read at a time
mov dl, <drive_num>     ; The external drive's ID
mov cl, <start_sector>  ; The sector to start reading from
mov dh, <head>          ; The head to read from
mov ch, <cylinder>      ; The cylinder to read from
int 0x13                ; Drive services interrupt
jc <error_handler>      ; Jump to error handler on CF set
```

RTC

```
mov ah, 0x00             ; Select 'Read RTC' function
int 0x1A                 ; RTC services interrupt
shl ecx, 16              ; Clock ticks are split in the CX:DX pair, so shift ECX left by 16...
or cx, dx                ; and add in the low half of the pair
mov <new_day>, al        ; AL is non-zero if the last call to this function was before
midnight
                         ; Now ECX holds the clock ticks (approx. 18.2/sec) since midnight
                         ; and <new_day> is non-zero if we passed midnight since the last read
```

RTC

```
mov ah, 0x02             ; Select 'Read system time' function
int 0x1A                 ; RTC services interrupt
                         ; Now CH contains hour, CL minutes, DH seconds, and DL the DST flag,
                         ; all encoded in BCD (DL is zero if in standard time)
                         ; Now we can decode them into a string (we'll ignore DST for now)

mov al, ch               ; Get hour
shr al, 4                ; Discard one's place for now
add al, 48               ; Add ASCII code of digit 0
mov [CLOCK_STRING+0], al ; Set ten's place of hour
mov al, ch               ; Get hour again
and al, 0x0F             ; Discard ten's place this time
add al, 48               ; Add ASCII code of digit 0 again
mov [CLOCK_STRING+1], al ; Set one's place of hour

mov al, cl               ; Get minute
shr al, 4                ; Discard one's place for now
add al, 48               ; Add ASCII code of digit 0
mov [CLOCK_STRING+3], al ; Set ten's place of minute
mov al, cl               ; Get minute again
and al, 0x0F             ; Discard ten's place this time
add al, 48               ; Add ASCII code of digit 0 again
mov [CLOCK_STRING+4], al ; Set one's place of minute

mov al, dh               ; Get second
shr al, 4                ; Discard one's place for now
add al, 48               ; Add ASCII code of digit 0
mov [CLOCK_STRING+6], al ; Set ten's place of second
mov al, dh               ; Get second again
and al, 0x0F             ; Discard ten's place this time
add al, 48               ; Add ASCII code of digit 0 again
mov [CLOCK_STRING+7], al ; Set one's place of second
...
db CLOCK_STRING "00:00:00", 0   ; Place in some separate (non-code) area
```

RTC

```
mov ah, 0x04             ; Select 'Read system date' function
int 0x1A                 ; RTC services interrupt
                         ; Now CH contains century, CL year, DH month, and DL day, all in BCD
                         ; Decoding to a string is similar to the RTC Time example above
```

```
int 0x12                ; Conventional memory interrupt (no function select parameter)
and eax, 0xFFFF         ; AX contains kilobytes of conventional memory; clear high bits of
EAX
shl eax, 10             ; Multiply by 1 kilobyte (1024 bytes = 2^10 bytes)
                        ; EAX contains the number of bytes available from address 0000:0000
```

```
int 0x19                ; That's it! One call. Just make sure nothing has overwritten the
                        ; interrupt vector table, since this call does NOT restore them to
the
                        ; default values of normal power-up. This means this call will not
                        ; work too well in an environment with an operating system loaded.
```

BIOS。 `AH0x860x80` 。 **CF**。 `jc`。

BIOSRalf Brown 。 HTML。

。

osdev.org

https://riptutorial.com/zh-TW/x86/topic/6946/

12:

Examples

16

80862016◦ 816 -

- AX◦
  ◦
- DX◦
  3216AX - ◦
- CX◦
  - LOOPNE REP /
- BX◦
  - ◦
- SI◦
  ◦
- DI◦
  ◦
- SP◦
  PUSHPOPCALLCALLRET ◦
- BP◦
  “”◦ BP◦

1. ◦
2. BX SIDI◦

   ```
   MOV    AX, [BX+5]     ; Point into Data Segment
   MOV    AX, ES:[DI+5]  ; Override into Extra Segment
   ```

3. DIMOVSCMPS ◦ ◦
4. SPBP◦

32

803861632◦ 323232◦ ◦ 1632 -

32◦ 81632E“” EAX EBX ECX EDX ESI EDI EBPESP ◦ 1632ADDCMP ◦ 8◦

1632 CMP AX,DX 16CMP AX,DXCMP EAX,EDX 32CMP EAX,EDX

“AX ”0xB8 0xB8 0x12 0x34

“EAX ”0xB8 0xB8 0x12 0x34 0x56 0x78

assember。

8

16

- AHALAX。
- BHBLBX。
- CHCLCX。
- DHDLDX。

AHALAX 8“” - AL0xFF0x00AH 。

648

- RSI SIL
- DIL for RDI
- RBP BPL
- RSP SPL

R8R15 R8B - R15B 。

8086168 - 16。 1665,536。

“” - 1664 - 。

- CS。
  IP 。
- DS。
  。
- ES。
  。
- SS。
  。

16。 。

32--4 - 8“”2416。 112 -

803864。

- FS
- GS

Segment。

C D E.....

64

AMD80386。 。

326432x8664。 64AMD。

64

- 64R RAX RBX RCX RDX RSI RDI RBPRSP。

  E。

- 864 R8 R9 R10 R11 R12 R13 R14R15。
  - 32R8DR15D DDWORD。
  - 16W R8WR15W。
- *168*
  - AL BL CLDL;
  - SIL DIL BPLSPL;
  - 8 R8BR15B。
  - AH BH CHDHREX64R8-R15SIL DIL BPLSPL。 REXAHSPL。 23-1。

3232816。 。

x86ALUNOTADD16FLAGS“”“”“”。 3232EFLAGS 6464RFLAGS。

-。 JccSETcc cc“”

E Z		ZF == 1
NE NZ		ZF == 0
O		OF == 1
NO		OF == 0
S		SF == 1
NS		SF == 0

P		PF == 1
NP		PF == 0
----------	----	----------
C B NAE		CF == 1
NC NB AE		CF == 0
A NBE		CF == 0ZF == 0
NA BE		CF == 1ZF == 1
----------	----	----------
GE NL		SF == OF
NGE L		SF = OF
G NLE		ZF == 0SF == OF
NG LE		ZF == 1SF = OF

160165,535-1 - 0xFFFF ◦ ◦ 0x8000132,76832,767 ;-32,76832,767 -

◦ “Above”“Below”“Greater”“Less”◦ JB“”JL“”◦

FLAGS

- LAHFAH
- SAHF AHFlags

◦ FLAGS / EFLAGS / RFLAGS

- PUSHF / POPF 16FLAGS/
- PUSHFD / POPFD 32EFLAGS/
- PUSHFQ / POPFQ PUSH / POP 64RFLAGS/

[R/E]FLAGS◦

ALU FLAGS

- IF◦

  STICLI◦
- DF◦

  CMPSMOVS ◦ DFCLD;STD◦
- TF◦ ◦ “”“”◦ ◦

80286

80286FLAGS

- IOPL I / O。
  I / O。 00 2 11 2。 IOPLI / O。
- NT。
  CALL。 RET。

80386

'386。

- RF。
  `386。 Resume Flag。 Resume FlagDebug。
- VM8086。
  163280386“8086”16Virtual 8086。 VM8086。

80486

。 。

- ACx8644。 。 AC。 AC set。

PentiumCPUID

- VIF。
  IF - 。
- VIP。
  VIFTaskSTI。
- ID CPUID -allowed。
  CPUID。 “”。

https://riptutorial.com/zh-TW/x86/topic/2122/

S. No		Contributors
1	x86	Community, David Hoelzer, Peter Cordes, Peter Mortensen, PyNEwbie, Runner
2		Cody Gray, Downvoter, faissaloo, John Burger, sannaj, Stephen Leppik
3	-	John Burger
4		Cody Gray, icktoofay, Margaret Bloom, Michael Petch, Peter Cordes, user45891, Zopesconk
5		Margaret Bloom, Michael Petch, RamenChef
6		Margaret Bloom, MikeCAT
7		John Burger
8		Margaret Bloom, owacoder, Ped7g, Zopesconk
9		Ped7g, Zopesconk
10		John Burger, Margaret Bloom
11		owacoder
12		hidefromkgb, John Burger, Ped7g, Peter Cordes