Intel x86 CPU architecture

Saulius Gražulis

Vilnius, 2020

Vilnius University, Faculty of Mathematics and Informatics
Institute of Informatics

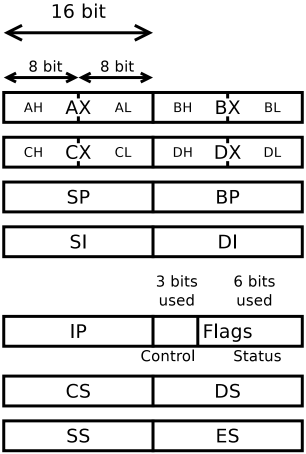
What is a processor architecture?

Architecture visible to programmer:

- The CPU registers visible to programmer
- Memory addressing
- Data formats
- Processor instruction set
- Input-output
- Interrupt processing

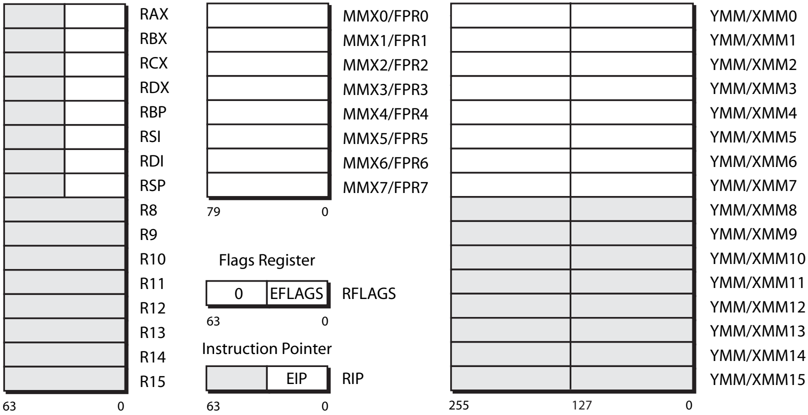
Registers (x86)

Intel 8086 CPU



(Intel 1979)

Registers (x86_64)



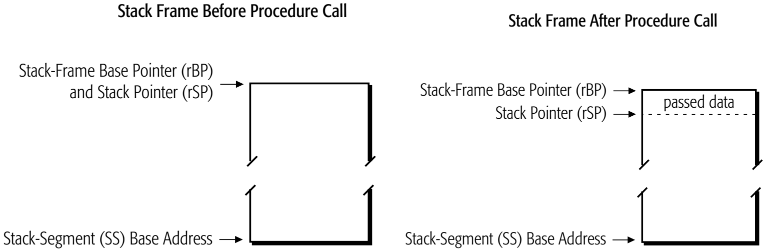
□ Legacy x86 registers, supported in all modes

▨ Register extensions, supported in 64-bit mode

Application-programming registers not shown include Media eXension Control and Status Register (MXCSR) and x87 tag-word, control-word, and status-word registers

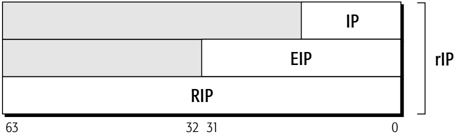
(AMD 2017)

Stack operation


Stack Frame Before Procedure Call
Stack-Frame Base Pointer (rBP)
and Stack Pointer (rSP)
Stack-Segment (SS) Base Address
Stack Frame After Procedure Call
Stack-Frame Base Pointer (rBP)
Stack Pointer (rSP)
passed data
Stack-Segment (SS) Base Address

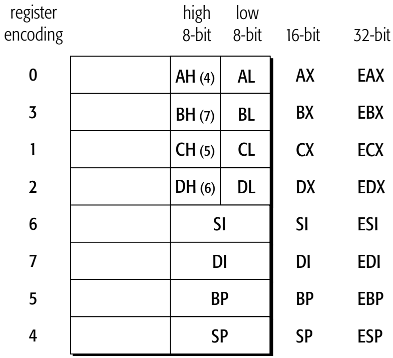
(AMD 2017)

Instruction pointer



(AMD 2017)

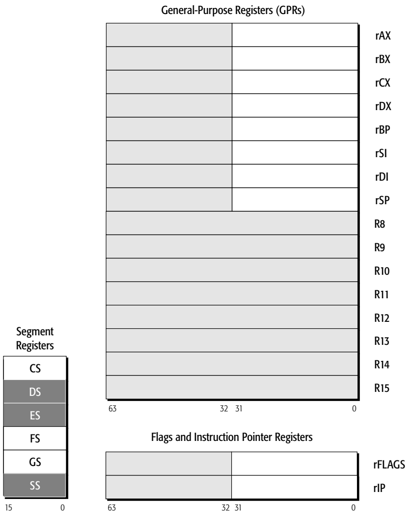
Registers, Legacy and Compatibility modes



register encoding		high 8-bit	low 8-bit	16-bit	32-bit
0		AH (4)	AL	AX	EAX
3		BH (7)	BL	BX	EBX
1		CH (5)	CL	CX	ECX
2		DH (6)	DL	DX	EDX
6		SI		SI	ESI
7		DI		DI	EDI
5		BP		BP	EBP
4		SP		SP	ESP

(AMD 2017)

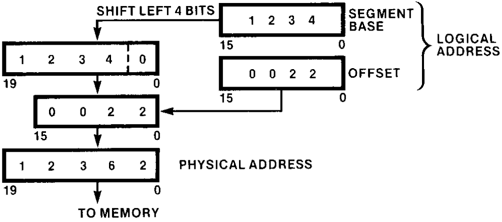
Registers



(AMD 2017)

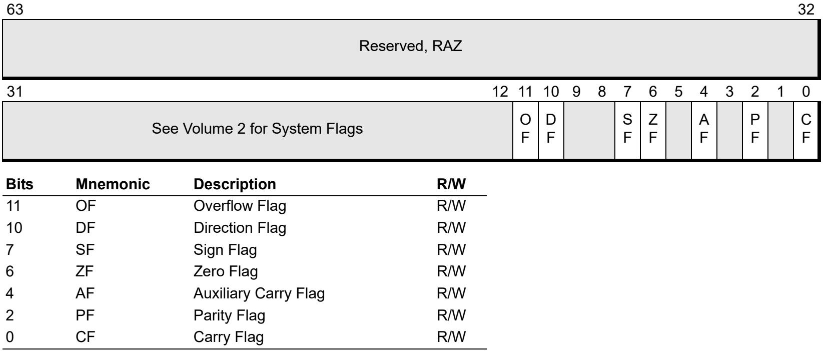
Segmented addressing

Segment address is shifted by 4 bits and added to the offset:



(Intel 1979)

Flag register



Bits	Mnemonic	Description	R/W
11	OF	Overflow Flag	R/W
10	DF	Direction Flag	R/W
7	SF	Sign Flag	R/W
6	ZF	Zero Flag	R/W
4	AF	Auxiliary Carry Flag	R/W
2	PF	Parity Flag	R/W
0	CF	Carry Flag	R/W

(AMD 2017)

Flags

- **Carry Flag (CF)** – 1 if the last integer addition or subtraction operation resulted in a carry (for addition) or a borrow (for subtraction) out of the most-significant bit position of the result. Increment and decrement instructions—unlike the addition and subtraction instructions—do not affect the carry flag.
- **Overflow Flag (OF)**
- **Auxiliary Overflow Flag (AF)**
- **Parity Flag (PF)** – 1 if there is an even number of 1 bits in the least-significant byte of the last result of certain operations.
- **Zero Flag (ZF)**
- **Sign Flag (SF)**
- **Directions Flag (DF)**

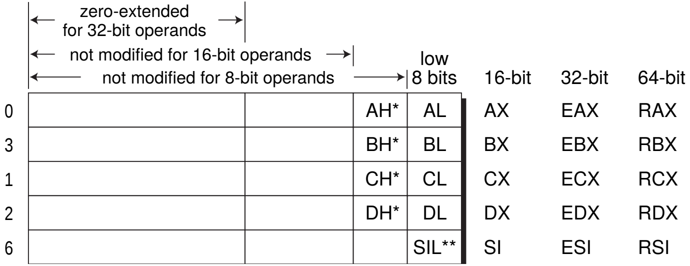
Instruction types

- Data Movement
- Conversion Instructions
- Arithmetic Instructions
- Logical Instructions
- Control Instructions

(Jorgensen 2020)

Zero extension of 32-bit operands

As Figure 3-3 on page 27 and Figure 3-4 on page 28 show, when performing 32-bit operations with a GPR destination in 64-bit mode, the processor zero-extends the 32-bit result into the full 64-bit destination (AMD 2017), p. 29.



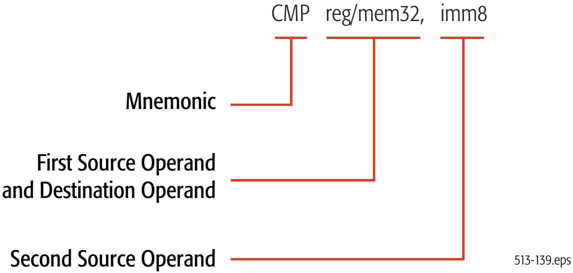
zero-extended for 32-bit operands

not modified for 16-bit operands

not modified for 8-bit operands

				low 8 bits	16-bit	32-bit	64-bit
0			AH*	AL	AX	EAX	RAX
3			BH*	BL	BX	EBX	RBX
1			CH*	CL	CX	ECX	RCX
2			DH*	DL	DX	EDX	RDX
6				SIL**	SI	ESI	RSI

Instruction syntax



(AMD 2017)

https://en.wikipedia.org/wiki/X86_instruction_listings

Number of instructions – x86

- To the assembly language programmer, the 8086 and 8088 appear to have a repertoire of about 100 instructions. (Intel 1979)
- The 8086 and 8088 CPUs, however, recognise 28 different MOV machine instructions ("move byte register to memory," "move word immediate to register," etc.).
    - … `MOV` is actually Turing complete! (Dolan 2013)
- The x86_64 architecture has roughly between 1000 and 3000 instructions, depending on how you count… [1]
- The x86_64 architecture *instruction space* (1-15 bytes) is $1.3 \cdot 10^{36}$ possible instructions. This can be reduced to a "very manageable" $\approx 10^{8}$ by a clever depth-first search (Domas 2017)

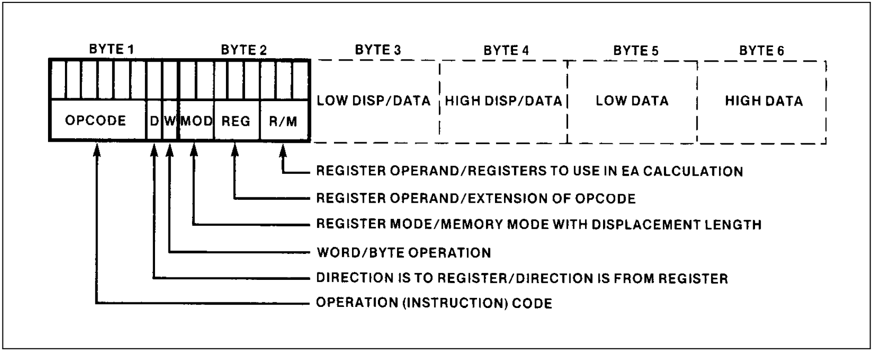
[1] https://stefanheule.com/blog/how-many-x86-64-instructions-are-there-anyway

Operand counts

- Zero-address machines: `OPCODE`
- One-address machines: `OPCODE` `OPERAND`
- Two-address machines: `OPCODE` `OP1` `OP2`
- Three-address machines: `OPCODE` `OP1` `OP2` `OP3`
- Four-address machines: `OPCODE` `OP1` `OP2` `OP3` `OP4`

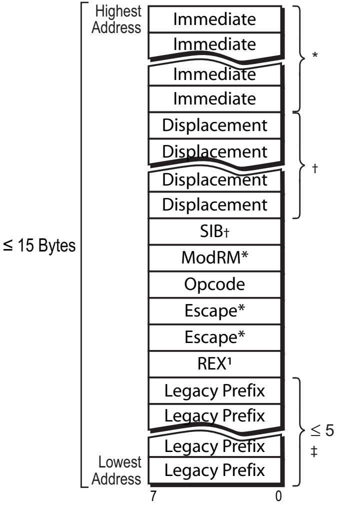
Instruction encoding – 8086

Only one memory operand can be present!


BYTE 1
BYTE 2
BYTE 3
BYTE 4
BYTE 5
BYTE 6
OPCODE
D
W
MOD
REG
R/M
LOW DISP/DATA
HIGH DISP/DATA
LOW DATA
HIGH DATA
REGISTER OPERAND/REGISTERS TO USE IN EA CALCULATION
REGISTER OPERAND/EXTENSION OF OPCODE
REGISTER MODE/MEMORY MODE WITH DISPLACEMENT LENGTH
WORD/BYTE OPERATION
DIRECTION IS TO REGISTER/DIRECTION IS FROM REGISTER
OPERATION (INSTRUCTION) CODE

Intel 8086 Family User's guide

Instruction encoding – 64 bit mode



(AMD 2017), Vol. 3

Register encoding – 8086

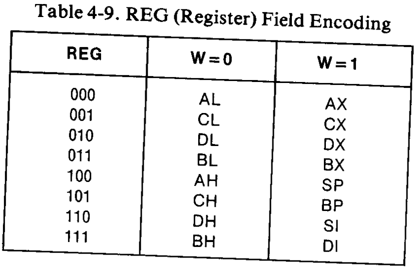


Table 4-9. REG (Register) Field Encoding

REG	W = 0	W = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Intel 8086 Family User's guide, p. 162

Mode and register encoding – 8086

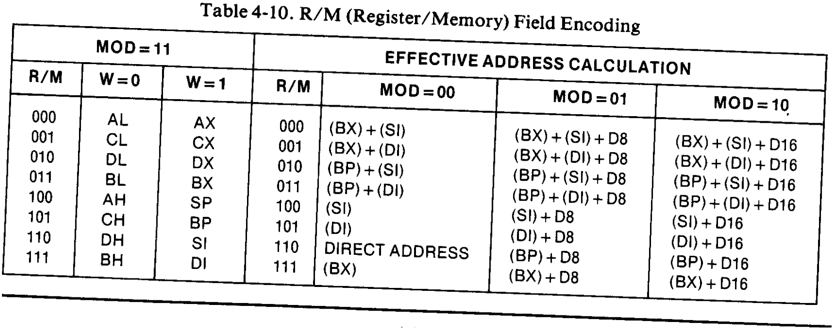


Table 4-10. R/M (Register/Memory) Field Encoding

MOD = 11			EFFECTIVE ADDRESS CALCULATION			
R/M	W = 0	W = 1	R/M	MOD = 00	MOD = 01	MOD = 10
000	AL	AX	000	(BX) + (SI)	(BX) + (SI) + D8	(BX) + (SI) + D16
001	CL	CX	001	(BX) + (DI)	(BX) + (DI) + D8	(BX) + (DI) + D16
010	DL	DX	010	(BP) + (SI)	(BP) + (SI) + D8	(BP) + (SI) + D16
011	BL	BX	011	(BP) + (DI)	(BP) + (DI) + D8	(BP) + (DI) + D16
100	AH	SP	100	(SI)	(SI) + D8	(SI) + D16
101	CH	BP	101	(DI)	(DI) + D8	(DI) + D16
110	DH	SI	110	DIRECT ADDRESS	(BP) + D8	(BP) + D16
111	BH	DI	111	(BX)	(BX) + D8	(BX) + D16

Intel 8086 Family User's guide, p. 162

MOV instruction

nasm:

```
     5 00000000 A0[2D00]                          mov al, [b1]
     6 00000003 8A26[2E00]                        mov ah, [b2]
     7 00000007 A1[2F00]                          mov ax, [w1]
     8
     9 0000000A 8A1E[2D00]                        mov bl, [b1]
    10 0000000E 8A3E[2E00]                        mov bh, [b2]
    11 00000012 8B1E[2F00]                        mov bx, [w1]
    12
    13 00000016 8B18                              mov bx, [bx + si]
    14 00000018 8B98[2F00]                        mov bx, [bx + si + w1]
    15
    16 0000001C 8B00                              mov ax, [bx + si]
    17 0000001E 8B803412                          mov ax, [bx + si + 1234h]
    18
    20 00000024 89C8                              mov ax, cx
    21 00000026 88C4                              mov ah, al
    22
```

LEA instruction

nasm:

```
 5 00000000 8D1A                                lea bx, [bp + si]
 6 00000002 8D9A[1300]                          lea bx, [bp + si + w1]
 7
 8 00000006 8D02                                lea ax, [bp + si]
 9 00000008 8D82[1300]                          lea ax, [bp + si + w1]
10
11                                              ;;  exit:
12 0000000C B8004C                              mov ax, 4c00h
13 0000000F CD21                                int 21h
14
15 00000011 55                          b1:     DB 055h
```

XCHG instruction

```
 5 00000000 8706[1000]                      xchg ax, [w1]
 6 00000004 8606[1200]                      xchg al, [b1]
 7
 8 00000008 92                              xchg ax, dx
 9 00000009 86D8                            xchg bl, al
10
11                                          ;;  exit:
15 00000010 2301                    w1:     DW 0123h
16 00000012 0100                    b1:     DW 01h
```

If a memory operand is referenced, the processor's locking protocol is automatically implemented for the duration of the exchange operation, regardless of the presence or absence of the LOCK prefix or of the value of the IOPL. (See the LOCK prefix description in this chapter for more information on the locking protocol.) This instruction is useful for implementing semaphores or similar data structures for process synchronization.

https://c9x.me/x86/html/file_module_x86_id_328.html

2020-03-23

ADD, ADC instructions

nasm:

```
 5 00000000 01D8                           add ax, bx
 6 00000002 11D1                           adc cx, dx
 7
 8 00000004 01F2                           add dx, si
 9 00000006 0314                           add dx, [si]
10
11 00000008 0316[1E00]                     add dx, [w1]
12 0000000C 0394[1E00]                     add dx, [si+w1]
13 00000010 03547F                         add dx, [si+07Fh]
14 00000013 03948000                       add dx, [si+080h]
15
20 0000001C 55                     b1:     DB 055h
21 0000001D AA                     b2:     DB 0AAh
22 0000001E 2301                   w1:     DW 0123h
```

INC, DEC instructions

```
     5 00000000 40                                    inc ax
     6 00000001 49                                    dec cx
     7
     8 00000002 FE04                                  inc byte [si]
     9 00000004 FF447F                                inc word [si+07Fh]
    10 00000007 FF888000                              dec word [si+bx+080h]
```

REP STOS instruction

```
 5 00000000 FC                                  cld
 6 00000001 B0AA                                mov al, 0AAh
 7 00000003 BF[1000]                            mov di, str
 8 00000006 B91200                              mov cx, strend - str
 9
10 00000009 F3AA                                rep stosb
11
16 00000010 412076657279206C6F-       str:      DB 'A very long srting'
16 00000019 6E6720737274696E67
17 00000022 00                         strend:   DB 0
```

LOOP instruction

```
     5 00000000 B90500                          mov cx, 5
     6 00000003 B80300                          mov ax, 3
     7 00000006 89C3                            mov bx, ax
     8 00000008 89C6                            mov si, ax
     9
    10 0000000A F7EB                    begin:  imul bx
    11 0000000C 0FAFF3                          imul si, bx
    12 0000000F E2F9                            loop begin
```

JMP,Jcc instructions

```
     5 00000000 E410                            in  al, 010h
     6
     7 00000002 3C00                            cmp al, 0
     8
     9 00000004 7A04                            jpe even
    10 00000006 B401                            mov ah, 1
    11 00000008 EB02                            jmp finish
    12 0000000A B402                    even:   mov ah, 2
    13                                  finish:
```

PUSH/POP instructions

```
     5 00000000 50                                     push ax
     6 00000001 56                                     push si
     7 00000002 1E                                     push ds
     8 00000003 6A0A                                   push 10
     9
    10 00000005 58                                     pop ax
    11 00000006 5B                                     pop bx
    12 00000007 59                                     pop cx
    13 00000008 5A                                     pop dx
```

CALL and RET instructions

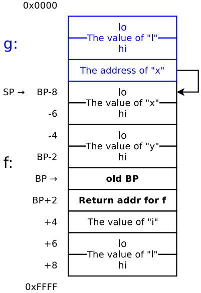
```
 5 00000000 E80500                           call subr
 6
11 00000008 B83412                    subr:  mov ax, 01234h
12 0000000B C3                               ret
```

Stack frame

```
int main( )
{
  static
  int a; long b;

  /* ... */
  a = f( a, b );
  /* ... */
  return 0;
}
```

```
mov ax, [b+2]
push ax
mov ax, [b]
push ax
mov ax, [a]
push ax
call f
add sp, 6
mov [a], ax
```



```
int f( int i, long l )
{
  float x, y;

  /* ... */
  i = g( l, &x );
  /* ... */
  return i;
}
```

```
push bp
mov bp, sp
sub sp, 8
; ...
lea ax, [bp-8]
push ax ; place &x onto the stack
mov ax, [bp+8] ; msw of "l"
push ax ; msw of "l" → stack
mov ax, [bp+6] ; lsw of "l"
push ax ; lsw of "l" → stack
call g
; ...
add sp, 8
pop bp
ret
```

SYSCALL/INT and SYSRET/IRET instructions

16-bit (nasm):

```
     7                                          ;;  exit:
     8 00000003 B8004C                          mov ax, 4c00h
     9 00000006 CD21                            int 21h
```

64-bit (yasm):

```
     8                                  EXIT_SUCCESS equ 0
     9                                  SYS_exit equ 60
    10                                  SYS_write equ 1
    11
    26 0000001F 48C7C03C000000          mov rax, SYS_exit
    27 00000026 48C7C700000000          mov rdi, EXIT_SUCCESS
    28 0000002D 0F05                    syscall
```

References

AMD (Dec. 2017). *AMD64 Architecture Programmer's Manual, Volume 1: Application Programming, revision 3.22*. AMD. URL: https://www.amd.com/system/files/TechDocs/24592.pdf.

Dolan, Stephen (2013). *mov is Turing-complete*. URL: https://drwho.virtadpt.net/files/mov.pdf.

Domas, Christopher (2017). *Breaking the x86 ISA*. URL: https://github.com/xoreaxeaxeax/sandsifter/blob/master/references/domas_breaking_the_x86_isa_wp.pdf.

Intel (Oct. 1979). *Intel 8086 Family User's Manual*. Intel Corporation. URL: https://edge.edx.org/c4x/BITSPilani/EEE231/asset/8086_family_Users_Manual_1_.pdf.

Jorgensen, Ed (Jan. 2020). *x86-64 Assembly Language Programming with Ubuntu*. Vol. 1. URL: http://www.egr.unlv.edu/ẽd/assembly64.pdf. Ed Jorgensen.