x86 Architecture Overview

The IA-32 is the instruction set architecture (ISA) of Intel's most successful line of 32-bit processors, and the Intel 64 ISA is its extension into 64-bit processors. (Actually Intel 64 was invented by AMD, who called it x86-64). These notes summarize a few items of interest about these two ISAs. They in no way serve as a substitute for reading Intel's manuals.

CONTENTS

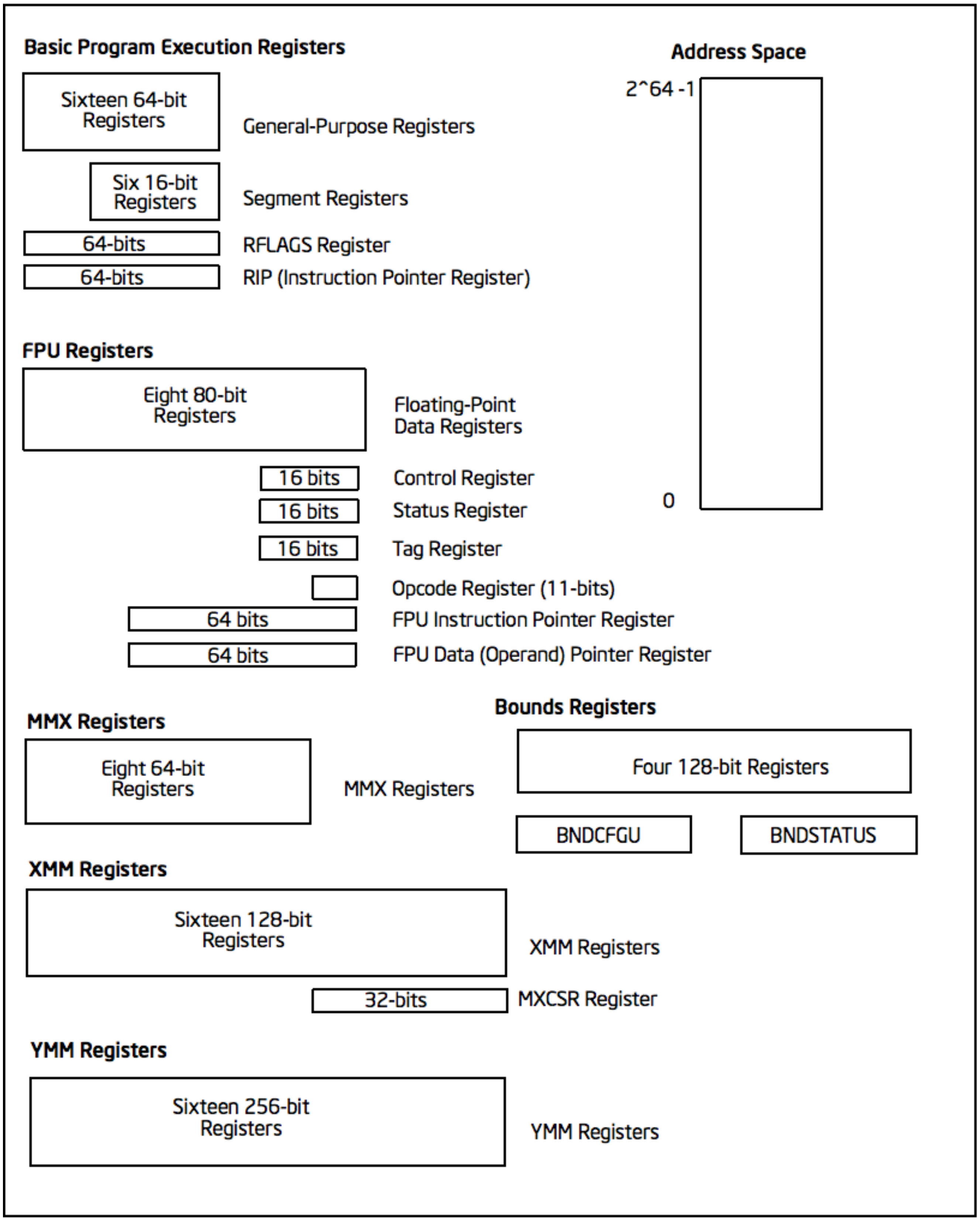
IA-32 and x86-64

The two massively popular architectures IA-32 and x86-84 are so common, they are described in a single set of manuals.

The following notes briefly summarize the latter architecture only.

x86-64 Architecture Diagram

The basic architecture of the x86-64 is described in Volume 1 of the System Developer's Manual. The following diagram is taken directly from Chapter 3 in this volume:


Basic Program Execution Registers
Sixteen 64-bit Registers
General-Purpose Registers
Six 16-bit Registers
Segment Registers
64-bits
RFLAGS Register
64-bits
RIP (Instruction Pointer Register)
Address Space
2^64 -1
0
FPU Registers
Eight 80-bit Registers
Floating-Point Data Registers
16 bits
Control Register
16 bits
Status Register
16 bits
Tag Register
Opcode Register (11-bits)
64 bits
FPU Instruction Pointer Register
64 bits
FPU Data (Operand) Pointer Register
MMX Registers
Eight 64-bit Registers
MMX Registers
Bounds Registers
Four 128-bit Registers
BNDCFGU
BNDSTATUS
XMM Registers
Sixteen 128-bit Registers
XMM Registers
32-bits
MXCSR Register
YMM Registers
Sixteen 256-bit Registers
YMM Registers

Registers

Application Programmers generally use only the general purpose registers, floating point registers, XMM, and YMM registers.

General Purpose Registers

These are 64 bits wide and used for integer arithmetic and logic, and to hold both data and pointers to memory. The registers are called R0...R15. Also:

- You can access the lower order 32-bits of each register using the names R0D...R15D. The “D” stands for “doubleword” because strangely, the word “word” on this platform refers to a 16-bit quantity. Why? Backward compatibilty! The x86-64 grew out of a 16-bit processor family created in the 1970s.
- You can access the lower order 16-bits of each register using the names R0W...R15W.
- You can access the lower order 8-bits of each register using the names R0B...R15B.
- R0...R7 have aliases RAX, RCX, RBX, RDX, RSP, RBP, RSI, RDI, respectively.
- R0D...R7D have aliases EAX, ECX, EBX, EDX, ESP, EBP, ESI, EDI, respectively.
- R0W...R7W have aliases AX, CX, BX, DX, SP, BP, SI, DI, respectively.
- R0B...R7B have aliases AL, CL, BL, DL, SPL, BPL, SIL, DIL, respectively.

RIP and RFLAGS

RIP is the instruction pointer and RFAGS is the flags register.

Segment Registers

These are CS, DS, SS, ES, FS, and GS. I haven’t used them in 64-bit programming.

XMM Registers

These are 128-bits wide. They are named XMM0...XMM15. Use them for floating-point and integer arithmetic. You can do operations on 128-bit integers, but you can also take advantage of their ability to do operations in parallel:

- Two 64-bit integer operations in parallel

- Four 32-bit integer operations in parallel
- Eight 16-bit integer operations in parallel
- Sixteen 8-bit integer operations in parallel
- Two 64-bit floating-point operations in parallel
- Four 32-bit floating-point operations in parallel

YMM Registers

These are 256-bits wide. They are named YMM0...YMM15. Use them for floating-point arithmetic. You can do:

- Four 64-bit floating-point operations in parallel
- Eight 32-bit floating-point operations in parallel

and some other crazy things.

FPU Registers

There are eight registers used for computing with 80-bit floating point values. The registers don't have names because they are used in a stack-like fashion.

Other Registers

Application programmers can remain oblivious of the rest of the registers:

- The 8 32-bit **processor control registers**: CR0, CR1, CR2, CR3, CR4, CR5, CR6, CR7. The lower 16 bits of CR0 is called the Machine Status Word (MSW).
- The 4 16-bit **table registers**: GDTR, IDTR, LDTR and TR.
- The 8 32-bit **debug registers**: DR0, DR1, DR2, DR3, DR4, DR5, DR6 and DR7.
- The 5 test registers: TR3, TR4, TR5, TR6 and TR7.
- The **memory type range registers**
- The **machine specific registers**
- The **machine check registers**

Instruction Set

See the SDM Volume 1, Chapter 5 for a nice overview of **all** of the processor instructions and Volume 2 for complete information.

The following table shows most of the available instructions, using the instruction names as specified in the Intel syntax. Not every processor supports every instruction, of course.

The vertical bar means OR, the square brackets mean OPTIONAL, and parentheses are used for grouping. For example:

- `SH(L|R)[D]` stands for `SHL`, `SHR`, `SHLD`, `SHRD`.
- `PUSH[A[D]]` stands for `PUSH`, `PUSHA`, `PUSHAD`.

INTEGER	FPU	SSE	SSE2
MOV	F[I]LD	MOV(A|U)PS	MOV(A|U)PD
CMOV[N]((L|G|A|B)[E]|E|Z|S|C|O|P)	F[I]ST[P]	MOV(H|HL|L|LH)PS	MOV(H|L)PD
XCHG	FBLD	MOVSS	MOVSD
BSWAP	FBSTP	MOVMSKPS	MOVMSKPD
XADD	FXCH		
CMPXCHG[8B]	FCMOV[N](E|B|BE|U)	ADD(P|S)S	ADD(P|S)D
PUSH[A[D]] | POP[A[D]]		SUB(P|S)S	SUB(P|S)D
IN | OUT	FADD[P]	MUL(P|S)S	MUL(P|S)D
CBW | CWDE | CWD | CDQ	FIADD	DIV(P|S)S	DIV(P|S)D
MOVSX | MOVZX	FSUB[R][P]	RCP(P|S)S	SQRT(P|S)D
	FISUB[R]	SQRT(P|S)S	MAX(P|S)D
ADD | ADC	FMUL[P]	RSQRT(P|S)S	MIN(P|S)D
SUB | SBB	FIMUL	MAX(P|S)S	
[I]MUL	FDIV[R][P]	MIN(P|S)S	CMP(P|S)D
[I]DIV	FIDIV[R]		[U]COMISD
INC | DEC	FPREM[1]	CMP(P|S)S	
NEG	FABS	[U]COMISS	ANDPD
CMP	FCHS		ANDNPD
	FRNDINT	ANDPS	ORPD
DAA | DAS	FSCALE	ANDNPS	XORPD
AAA | AAS | AAM | AAD	FSQRT	ORPS	
	FXTRACT	XORPS	SHUFPD
AND | OR | XOR | NOT			UNPCK(H|L)PD
	F[U]COM[P][P]	SHUFPS	
SH(L|R)[D]	FICOM[P]	UNPCK(H|L)PS	CVT(PI|DQ)2PD
SA(L|R)	F[U]COMI[P]		CVT[T]PD2(PI|DQ)
RO(L|R)	FTST	CVTPI2PS	CVTSI2SD
RC(L|R)	FXAM	CVT[T]PS2PI	CVT[T]SD2SI
		CVTSI2SS	CVTPS2PD
BT[S|R|C]	FSIN	CVT[T]SS2SI	CVTPD2PS
BS(F|R)	FCOS		CVTDQ2PS
SET[N]((L|G|A|B)[E]|E|Z|S|C|O|P)	FSINCOS	PAVG(B|W)	CVT[T]PS2DQ
TEST	FPTAN	PEXTRW	CVTSS2SD
	FPATAN	PINSRW	CVTSD2SS
JMP	F2XM1	P(MIN|MAX)(UB|SW)	
J[N]((L|G|A|B)[E]|E|Z|S|C|O|P)	FYL2X	PMOVMSKB	MOVDQ(A|U)
J[E]CXZ	FYL2XP1	PMULHUW	MOVQ2DQ
LOOP[N][Z|E]		PSADBW	MOVDQ2Q
CALL | RET	FLD1	PSHUFW	PUNPCK(H|L)QDQ
INT[O] | IRET	FLDZ		PADDQ
ENTER | LEAVE	FLDPI	LDMXCSR	PSUBQ

BOUND MOVS[B|W|D] CMPS[B|W|D] SCAS[B|W|D] LODS[B|W|D] STOS[B|W|D] INS[B|W|D] OUTS[B|W|D] REP[N][Z|E] STC | CLC | CMC STD | CLD STI | CLI LAHF | SAHF PUSHF[D] | POPF[D] LDS | LES | LFS | LGS | LSS LEA NOP UD2 XLAT[B] CPUID	FLDL2E FLDLN2 FLDL2T FLDLG2 FINCSTP FDECSTP FFREE F[N]INIT F[N]CLEX F[N]STCW FLDCW F[N]STENV FLDENV F[N]SAVE FRSTOR F[N]STSW FWAIT | WAIT FNOP FXSAVE FXRSTOR	STMXCSR MASKMOVQ MOVNT(Q|PS) PREFETCHT(0|1|2) PREFETCHNTA SFENCE	PMULUDQ PSHUF(LW|HW|D) PS(L|R)LDQ MASKMOVDQU MOVNT(PD|DQ|I) CLFLUSH LFENCE MFENCE PAUSE
SYSTEM	**MMX**	**SSE3**	**SSE4**
LGDT | SGDT LLDT | SLDT LTR | STR LIDT | SIDT LMSW | SMSW CLTS ARPL LAR LSL VERR | VERW INVD | WBINVD INVLPG LOCK HLT RSM RDMSR | WRMSR RDPMC RDTSC SYSENTER SYSEXIT	MOVD MOVQ PACKSS(WB|DW) PACKUSWB PUNPCK(H|L)(BW|WD|DQ) PADD(B|W|D) PADD(S|US)(B|W) PSUB(B|W|D) PSUB(S|US)(B|W) PMUL(H|L)W PMADDWD PCMP(EQ|GT)(B|W|D) PAND PANDN POR PXOR PS(L|R)L(W|D|Q) PSRA(W|D) EMMS	FISTTP LDDQU ADDSUBP(S|D) HADDP(S|D) HSUBP(S|D) MOVS(H|L)DUP MOVDDUP MONITOR MWAIT	PMUL(LD|DQ) DPP(D|S) MOVNTDQA BLEND[V](PD|PS) PBLEND(VB|W) PMIN(UW|UD|SB|SD) PMAX(UW|UD|SB|SD) ROUND(P|S)(S|D) EXTRACTPS INSERTPS PINSR(B|D|Q) PEXTR(B|W|D|Q) PMOV(S|Z)X(BW|BD|WD|BQ|WQ|DQ) MPSADBW PHMINPOSUW PTEST PCMPEQQ PACKUSDW PCMP(E|I)STR(I|M) PCMPGTQ CRC32 POPCNT
64-BIT MODE	**VIRTUAL MACHINE**	**SSSE3**	**AESNI**

CDQE
CMPSQ
CMPXCHG16B
LODSQ
MOVSQ
MOVZX
STOSQ
SWAPGS
SYSCALL
	VMPTRLD
	VPTRST
	VMCLEAR
	VMREAD
	VMWRITE
	VMCALL
	VMLAUNCH
	VMRESUME
	VMXOFF
		PHADD(W|SW|D)
		PHSUB(W|SW|D)
		PABS(B|W|D)
		PMADDUBSW
		PMULHRSW
		PSHUFB
		PSIGN(B|W|D)
		PALIGNR
			AESDEC[LAST]
			AESENC[LAST]
			AESIMC
			AESKEYGENASSIST
			PCLMULQDQ

`SYSRET`	`VMXON`
	`INVEPT`
	`INVVPID`		

Addressing Memory

In protected mode, applications can choose a flat or segmented memory model (see the SDM Volume 1, Chapter 3 for details); in real mode only a 16-bit segmented model is available. Most programmers will only use protected mode and a flat-memory model, so that's all we'll discuss here.

A memory reference has four parts and is often written as

```
[SELECTOR : BASE + INDEX * SCALE + OFFSET]
```

The selector is one of the six segment registers; the base is one of the eight general purpose registers; the index is any of the general purpose registers except ESP; the scale is 1, 2, 4, or 8; and the offset is any 32-bit number. (Example: `[fs:ecx+esi*8+93221]`.) The minimal reference consists of only a base register or only an offset; a scale can only appear if there is an index present.

Sometimes the memory reference is written like this:

```
selector
offset(base,index,scale)
```

Data Types

The data types are

Type name	Number of bits	Bit indices
Byte	8	7..0
Word	16	15..0
Doubleword	32	32..0
Quadword	64	63..0

Doublequadword	128	127..0

Little Endianness

The IA-32 is little endian, meaning the least significant bytes come first in memory. For example:

```
0   12
1   31        byte @ 9 = 1F
2   CB        word @ B = FE06
3   74        word @ 6 = 230B
4   67        word @ 1 = CB31
5   45        dword @ A = 7AFE0636
6   0B        qword @ 6 = 7AFE06361FA4230B
7   23        word @ 2 = 74CB
8   A4        qword @ 3 = 361FA4230B456774
9   1F        dword @ 9 = FE06361F
A   36
B   06
C   FE
D   7A
E   12
```

Note that if you draw memory with the lowest bytes at the bottom, then it is easier to read these values!

Flags Register

Many instructions cause the flags register to be updated. For example if you execute an `add` instruction and the sum is too big to fit into the destination register, the `Overflow` flag is set.

```
3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
```

```
 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+---------------------------------------------------------------+
| | | | | | | | | | |I|V|V|A|V|R| |N| I |O|D|I|T|S|Z| |A| |P| |C|
| | | | | | | | | | |D|I|I|C|M|F| |T| P |F|F|F|F|F|F| |F| |F| |F|
| | | | | | | | | | | |P|F| | | | | | L | | | | | | | | | | | | |
+---------------------------------------------------------------+
```

The flags are described in Section 3.4.3 of Volume 1 of the SDM. To determine how each instruction affects the flags, see Appendix A of Volume 1 of the SDM.

The System Developer's Manual

The System Developer's Manual contains vast amounts of important information and is required reading for all assembly language programmers and backend compiler writers. The manual is split into several volumes; links to all volumes are here. Highlights from Volumes 1 and 2:

- **Volume 1:**
  - Chapter 1 - About this manual
  - Chapter 2 - History of the IA-32 and Intel 64 architectures, a description of many of the microarchitectures and processors and technologies
  - Chapter 3 - Basic execution environment
  - Chapter 4 - Data types
  - Chapter 5 - Instruction set summary. Lists all instructions and a brief (but not precise) description of each. Instructions are grouped into convenient categories.
  - Chapter 6 - Details on calls and returns, and exceptions
  - Chapter 7 - All about general purpose instructions
  - Chapter 8 - All about FPU instructions
  - Chapter 9 - All about MMX instructions
  - Chapter 10 - All about SSE instructions
  - Chapter 11 - All about SSE2 instructions
  - Chapter 12 - All about SSE3, SSSE3, SSE4 and AESNI instructions
  - Chapter 13 - XSAVE
  - Chapter 14 - All about AVX, FMA, and AVX2 instructions

  - Chapter 15 - AVX-512
  - Chapter 16 - All about transactional synchronization instructions
  - Chapter 17 - Memory protection extensions
  - Chapter 18 - All about I/O instructions
  - Chapter 19 - How to determine what processor you have, and what its features are
  - Appendix A - Shows which instructions affect which flags
  - Appendix B - Condition codes
  - Appendix C - Floating-point exceptions
  - Appendix D - Guidelines for writing x87 exception handlers
  - Appendix E - Guidelines for writing SIMD exception handlers
- **Volume 2:**
  - Chapter 1 - About this manual
  - Chapter 2 - Instruction formats
  - Chapter 3-5 - Instruction set reference: full description, and encodings, of *every* instruction with names beginning with A — L
  - Chapter 6 - Safer Mode Extensions Reference
  - Appendix A - Opcode map
  - Appendix B - Encoding summary
  - Appendix C - Compiler intrinsics