WORDWARE APPLICATIONS LIBRARY

32/64-BIT 80x86 ASSEMBLY LANGUAGE ARC

LIBRARY OF CONGRESS
0 014 296 937 3

The increasing complexity of programming environments provides a number of opportunities for assembly language programmers. *32/64-Bit 80x86 Assembly Language Architecture* attempts to break through that complexity by providing a step-by-step understanding of programming Intel and AMD 80x86 processors in assembly language. This book explains 32-bit and 64-bit 80x86 assembly language programming inclusive of the SIMD (single instruction multiple data) instruction supersets that bring the 80x86 processor into the realm of the supercomputer, gives insight into the FPU (floating-point unit) chip in every Pentium processor, and offers strategies for optimizing code.

Learn about:

- 3DNow! MMX, SSE, SSE-2, SSE-3, AMD64, and EM64T instruction sets
- The similarities of and differences between various 80x86 processors
- Boolean bit manipulation
- Data swizzling, shuffling, and splatting
- Integer and floating-point math operations
- Branching and branchless coding methods
- Coding standards
- Debugging functions

About the author:

James Leiterman is a senior computer engineer who has been developing hardware and software professionally since 1978 and programming video/computer games for 20 years. He has worked for multiple game companies including Midway, LucasArts, Atari, Wild Goose Games, a casino gaming company, and two Internet online game companies and has shipped nine games. He is also the author of *Vector Game Math Processors* and *Learn Vertex and Pixel Shader Programming with DirectX 9*. He lives in Thousand Oaks, prospecting for California gold.

The companion files can be downloaded from www.wordware.com/files/8086 and www.leiterman.com/books.html.

Level: Intermediate to advanced
Category: Assembly language programming

ISBN-13: 978-1-59822-002-5
ISBN-10: 1-59822-002-0

$49.95 U.S. $69.95 Canada

9 781598 220025 54995

WORDWARE Publishing, Inc.
www.wordware.com

0 50826 04995 0 00020

Triplane illustration by Kenneth Mayfield. Cover design by Alan McCuller.

32/64-BIT 80x86 ASSEMBLY LANGUAGE ARCHITECTURE

QA 76
.9
.A73
L47
2005
Copy 2

FT MEADE
GenColl

WORDWARE APPLICATIONS LIBRARY

32/64-BIT 80x86 ASSEMBLY LANGUAGE ARCHITECTURE

JAMES C. LEITERMAN

WORDWARE Publishing, Inc.

32/64-Bit 80x86 Assembly Language Architecture

James C. Leiterman

Wordware Publishing, Inc.

Qualcomm Ex. 1050 - 2/25

QA76
.9
.A73L47
2005
Copy2

Library of Congress Cataloging-in-Publication Data

Leiterman, James (James C.)
32/64-bit 80x86 assembly language architecture / by James C. Leiterman.
p. cm.
Includes index.
ISBN-13: 978-1-59822-002-5
ISBN-10: 1-59822-002-0 (pbk.)
1. Computer architecture. 2. Assembler language (Computer program language) I. Title.
QA76.9.A73L47 2005
004.2'2--dc22

2005013995
CIP

© 2005, Wordware Publishing, Inc.

All Rights Reserved

2320 Los Rios Boulevard
Plano, Texas 75074

No part of this book may be reproduced in any form or by
any means without permission in writing from
Wordware Publishing, Inc.

Printed in the United States of America

ISBN-13: 978-1-59822-002-5
ISBN-10: 1-59822-002-0
10 9 8 7 6 5 4 3 2 1
0506

Brand names and product names mentioned in this book are trademarks or service marks of their respective companies. Any omission or misuse (of any kind) of service marks or trademarks should not be regarded as intent to infringe on the property of others. The publisher recognizes and respects all marks used by companies, manufacturers, and developers as a means to distinguish their products.

This book is sold as is, without warranty of any kind, either express or implied, respecting the contents of this book and any disks or programs that may accompany it, including but not limited to implied warranties for the book's quality, performance, merchantability, or fitness for any particular purpose. Neither Wordware Publishing, Inc. nor its dealers or distributors shall be liable to the purchaser or any other person or entity with respect to any liability, loss, or damage caused or alleged to have been caused directly or indirectly by this book.

All inquiries for volume purchases of this book should be addressed to Wordware Publishing, Inc., at the above address. Telephone inquiries may be made by calling:

(972) 423-0090

Qualcomm Ex. 1050 - 3/25

Contents

Preface xix

Chapter 1 **Introduction** **1**
Conventions Used in This Book 6
Companion Code 6
Image Patterning 7
Processor Legend 7
Notes, Tips, and Hints 9
Pseudo Vec 10
Pseudo Vec (x86) (3DNow!) (3DNow!+) (MMX) (MMX+) (SSE) (SSE2) (SSE3) 11
Graphics 101 (x86) (3DNow!) (MMX) 11
Algebraic Law 11
I-VU-Q 12

Chapter 2 **Coding Standards** **15**
Constants 16
Data Alignment 17
Stacks and Vectors 20
3D Vector (Floating-Point) 21
3D Quad Vector (Floating-Point) 21
Compiler Data 24
Assertions 25
Memory Systems 29
RamTest Memory Alignment Test 29
Memory Header 31
Allocate Memory (Malloc Wrapper) 31
Release Memory (Free Wrapper) 32
Allocate Memory 33
Allocate (Cleared) Memory 33
Free Memory — Pointer Is Set to NULL 34
Exercises 34

Chapter 3 **Processor Differential Insight** **35**
Processor Overview 35
History 38
The 64-Bit Processor 40
80x86 Registers 42
General-Purpose Registers 42

Qualcomm Ex. 1050 - 4/25

REX . . . 45
Segment/Selector Registers . . . 47
MMX Registers . . . 48
XMM Registers . . . 49
CPU Status Registers (EFLAGS/64-Bit RFLAGS) . . . 49
LAHF — Load AH Flags . . . 50
SAHF — Save AH Flags . . . 51
PUSHF/PUSHFD — Push EFLAGS onto Stack . . . 51
PUSHFQ — Push RFLAGS onto Stack . . . 51
POPF/POPFD — Pop EFLAGS from Stack . . . 52
POPFQ — Pop RFLAGS from Stack . . . 52
CLC — Clear (Reset) Carry Flag . . . 53
STC — Set Carry Flag . . . 54
CMC — Complement Carry Flag . . . 54
NOP — No Operation . . . 54
Floating-Point 101 . . . 55
Processor Data Type Encoding . . . 56
EMMS — Enter/Leave MMX State . . . 58
FEMMS — Enter/Leave MMX State . . . 58
Destination/Source Orientations . . . 59
Big/Little-Endian . . . 59
Alignment Quickie . . . 61
(Un)aligned Memory Access . . . 62
MOV/MOVQ — Move Data . . . 62
Move (Unaligned) . . . 62
Move (Aligned) . . . 63
Misaligned SSE(2) (128-bit) . . . 65
Aligned SSE(2) (128-bit) . . . 65
XCHG — Exchange (Swap) Data . . . 66
System Level Functionality . . . 67
Indirect Memory Addressing . . . 68
uint32 OddTable[ ] . . . 70
LEA — Load Effective Address . . . 70
Translation Table . . . 71
XLAT/XLATB — Translation Table Lookup . . . 71
String Instructions . . . 73
LODSB/LODSW/LODSD/LODSQ — Load String . . . 73
REP LODSx . . . 74
STOSB/STOSW/STOSD/STOSQ — Save String . . . 75
REP/REPE/REPZ/REPNE/REPNZ — Repeat String . . . 75
REP STOSx . . . 76
MOVSB/MOVSW/MOVSD/MOVSQ — Move String . . . 76
REP MOVSx . . . 77
CLD/STD — Clear/Set Direction Flag . . . 77
Special (Non-Temporal) Memory Instructions . . . 78
MOVNTx — Copy Using Non-Temporal Hint . . . 78
MOVNTPS — Copy 4×SPFP Using Non-Temporal Hint . . . 79

Qualcomm Ex. 1050 - 5/25

MOVNTPD — Copy 2×DPFP Using Non-Temporal Hint 79
MASKMOVQ/MASKMOVDQU — Copy Selected Bytes 80
Exercises 80

Chapter 4 Bit Mangling 81
Boolean Logical AND 82
Pseudo Vec 84
Logical Packed AND D=(A∧B) 84
vmp_pand (x86) 32-bit 85
vmp_pand (x86) 64-bit 86
vmp_pand (MMX) 86
vmp_pand (SSE2) Aligned Memory 86
vmp_pand (SSE2) Unaligned Memory 87
Boolean Logical OR 88
Pseudo Vec 89
Logical Packed OR D=(A∨B) 89
Boolean Logical XOR (Exclusive OR) 90
Pseudo Vec 91
Logical Packed XOR D=(A⊕B) 92
NOT — One's Complement Negation 92
NEG — Two's Complement Negation 93
ToolBox Snippet — The Butterfly Switch 94
I-VU-Q 96
Boolean Logical ANDC 99
Pseudo Vec 100
Logical Packed ANDC D=(A∧B′) 100
Exercises 101

Chapter 5 Bit Wrangling 103
Logical Left Shifting 103
SHL/SAL – Shift (Logical/Arithmetic) Left 104
SHLD — Shift (Logical) Left (Double) 105
PSLLx — Parallel Shift Left (Logical) 106
Pseudo Vec 107
Packed Shift Left Logical 16×8-bit by n:{0...7} 108
Pseudo Vec (x86) 109
vmp_psllB (MMX) 16×8-bit Vector 109
Logical Right Shifting 110
SHR — Shift (Logical) Right 110
SHRD — Shift (Logical) Right (Double) 111
PSRLx — Parallel Shift Right (Logical) 112
Pseudo Vec 113
Packed Shift Right Logical 16×8-bit by n:{0...7} 113
Arithmetic Right Shifting 114
SAR — Shift (Arithmetic) Right 114
PSRAx — Packed Shift Right (Arithmetic) 116
Pseudo Vec 117
Rotate Left (or n-Right) 117

Qualcomm Ex. 1050 - 6/25

ROL — Rotate Left 117
RCL — Rotate Carry Left 118
Rotate Right 120
ROR — Rotate Right 120
RCR — Rotate Carry Right 121
Bit Scanning 122
BSF — Bit Scan Forward 122
BSR — Bit Scan Reverse 122
ToolBox Snippet — Get Bit Count 123
Graphics 101 — Palette Bits 124
Exercises 127

Chapter 6 **Data Conversion 129**
Data Interlacing, Exchanging, Unpacking, and Merging 129
Byte Swapping 131
Little-Endian 132
(Big/Little)-Endian to (Big/Little)-Endian Data Relationship Macros 133
BSWAP — Byte Swap 133
PSWAPD — Packed Swap Double Word 135
Data Interlacing 136
PUNPCKLBW — Parallel Extend Lower from Byte 136
PUNPCKHBW — Parallel Extend Upper from Byte 137
PUNPCKLWD — Parallel Extend Lower from 16-Bit 137
PUNPCKHWD — Parallel Extend Upper from 16-Bit 138
PUNPCKLDQ — Parallel Extend Lower from 32-Bit 139
Also: (Unpack and Interleave Low Packed SPFP) 139
PUNPCKHDQ — Parallel Extend Upper from 32-Bit 139
ALSO: (Unpack and Interleave High Packed SPFP) 139
MOVSS — Move Scalar (SPFP) 140
MOVQ2DQ — Move Scalar (1×32-Bit) MMX to XMM 141
MOVDQ2Q — Move Scalar (1×32-bit) XMM to MMX 141
MOVLPS — Move Low Packed (2×SPFP) 142
MOVHPS — Move High Packed (2×SPFP) 143
MOVLHPS — Move Low to High Packed (2×SPFP) 143
MOVHLPS — Move High to Low Packed (2×SPFP) 144
MOVSD — Move Scalar (1×DPFP) 145
MOVLPD — Move Low Packed (1×DPFP) 146
MOVHPD — Move High Packed (1×DPFP) 147
PUNPCKLQDQ — Parallel Copy Lower (2×64-Bit) 147
Also: (Unpack and Interleave Low Packed Double-Precision Floating-Point Values) 147
PUNPCKHQDQ — Parallel Copy Upper (2×64-Bit) 148
Also: (Unpack and Interleave High Packed Double-Precision Floating-Point Values) 148
Swizzle, Shuffle, and Splat 149
PINSRW — Shuffle (1×16-Bit) to (4×16-Bit) 149
PSHUFW — Shuffle Packed Words (4×16-Bit) 150

Qualcomm Ex. 1050 - 7/25

PSHUFLW — Shuffle Packed Low Words (4×16-Bit). 151
PSHUFHW — Shuffle Packed High Words (4×16-Bit) 151
PSHUFD — Shuffle Packed Double Words (4×32-Bit) 152
SHUFPS — Shuffle Packed SPFP Values (4×SPFP). 153
MOVSLDUP — Splat Packed Even SPFP to (4×SPFP) 153
MOVSHDUP — Splat Packed Odd SPFP to (4×SPFP) 154
MOVDDUP — Splat Lower DPFP to Packed (2×DPFP) 154
SHUFPD — Shuffle Packed DPFP (2×64-Bit). 155
Data Bit Expansion 155
CBW Convert Signed AL (Byte) to AX (Word) 156
CWDE Convert Signed AX (Word) to EAX (DWord). 156
CDQE Convert Signed EAX (DWord) to RAX (QWord) 156
MOVSX/MOVSXD — Move with Sign Extension 157
MOVZX — Move with Zero Extension 158
CWD — Convert Signed AX (Word) to DX:AX. 160
CDQ — Convert Signed EAX (DWord) to EDX:EAX 160
CQO — Convert Signed RAX (QWord) to RDX:RAX 160
PEXTRW — Extract (4×16-bit) into Integer to (1×16) 161
Data Bit Reduction (with Saturation) 162
PACKSSWB — Packed Signed int16 to int8 with Saturation 162
PACKUSWB — Packed uint16 to uint8 with Saturation 162
PACKSSDW — Packed int32 to int16 with Saturation 163
Data Conversion (Integer : Float, Float : Integer, Float : Float). 164
PI2FW — Convert Packed Even int16 to SPFP 164
CVTDQ2PS — Convert Packed int32 to SPFP 165
CVTPS2DQ — Convert Packed SPFP to int32. 165
CVTPI2PS — Convert Lo Packed int32 to SPFP 166
CVTPS2PI — Convert Lo Packed SPFP to int32 166
CVTSI2SS — Convert Scalar int32 to SPFP. 167
CVTDQ2PD — Convert Even Packed int32 to DPFP 167
CVTPD2DQ — Convert Packed DPFP to Even int32 168
CVTPD2PS — Convert Packed DPFP to Lo SPFP 168
CVTPS2PD — Convert Lo Packed SPFP to DPFP 169
CVTPD2PI — Convert Packed DPFP to int32 169
CVTPI2PD — Convert Packed int32 to DPFP. 170
CVTSS2SI — Convert Scalar SPFP to int32/64 170
CVTSD2SI — Convert Scalar DPFP to Int. 171
CVTSI2SD — Convert Scalar Int to DPFP. 171
CVTSD2SS — Convert Scalar DPFP to SPFP 172
CVTSS2SD — Convert Scalar SPFP to DPFP 172
Exercises 172

Chapter 7 **Integer Math 175**
General Integer Math 175
ADD — Add. 176
ADC — Add with Carry 176
INC — Increment by 1 177
Thread-Safe Increment 179

Qualcomm Ex. 1050 - 8/25

XADD — Exchange and Add . . . 179
SUB — Subtract . . . 180
SBB — Subtract with Borrow . . . 180
DEC — Decrement by 1 . . . 181
Thread-Safe Decrement . . . 182
Packed Addition and Subtraction . . . 183
PADDB/PADDW/PADDD/PADDQ Integer Addition . . . 184
Vector {8/16/32/64}-Bit Int Addition with Saturation . . . 186
PSUBB/PSUBW/PSUBD/PSUBQ Integer Subtraction . . . 187
Vector {8/16/32/64}-Bit Integer Subtraction with Saturation . . . 188
Vector Addition and Subtraction (Fixed Point) . . . 189
Pseudo Vec . . . 189
Pseudo Vec (x86) . . . 190
vmp_paddB (MMX) 16×8-Bit . . . 190
vmp_paddB (SSE2) 16×8-Bit . . . 190
Averages . . . 191
PAVGB/PAVGUSB — N×8-Bit [Un]signed Integer Average . . . 191
PAVGW — N×16-Bit [Un]signed Integer Average . . . 192
Sum of Absolute Differences . . . 193
PSADBW — N×8-Bit Sum of Absolute Differences . . . 193
Integer Multiplication . . . 194
MUL — Unsigned Muliplication (Scalar) . . . 194
IMUL — Signed Multiplication (Scalar) . . . 194
Packed Integer Multiplication . . . 196
PMULLW — N×16-Bit Parallel Multiplication (Lower) . . . 196
PMULHW/PMULHUW — N×16-Bit Parallel Multiplication (Upper) . . . 198
PMULHRW — Signed 4×16-Bit Multiplication with Rounding (Upper) . . . 199
Pseudo Vec (x86) . . . 200
(MMX Mul Low) 4×16-Bit . . . 200
(SSE2 Mul Low) 8×16-Bit . . . 201
PMULUDQ — Unsigned N×32-Bit Multiply Even . . . 201
PMADDWD — Signed N×16-Bit Parallel Multiplication — ADD . . . 202
Integer Division . . . 204
DIV — Unsigned Division . . . 204
IDIV — Signed Division . . . 205
Exercises . . . 205

Chapter 8 **Floating-Point Anyone? . . . 207**
The Floating-Point Number . . . 208
FPU Registers . . . 212
Loading/Storing Numbers and the FPU Stack . . . 214
FLD — Floating-Point Load . . . 214
FST/FSTP — FPU Floating-Point Save . . . 217
FILD — FPU Integer Load . . . 217
FIST/FISTP/FISTTP — FPU Integer Save . . . 217
FPU Constants . . . 218

Qualcomm Ex. 1050 - 9/25

FXCH . . . 218
FINCSTP — FPU Increment Stack Pointer. . . . 219
FDECSTP — FPU Decrement Stack Pointer . . . 219
FWAIT/WAIT . . . 220
EMMS/FEMMS . . . 220
FNOP . . . 220
General Math Instructions . . . 221
FCHS — FPU Two's Complement ST(0) = – ST(0) . . . 221
FABS — FPU Absolute Value ST(0) = |ST(0)| . . . 221
FADD/FADDP/FIADD — FPU Addition D = ST(0) + A . . . 221
FSUB/FSUBP/FISUB — FPU Subtraction D = ST(0) – A . . . 222
FSUBR/FSUBRP/FISUBR — FPU Reverse Subtraction D = A – ST(0) . . . 223
FMUL/FMULP/FIMUL — FPU Multiplication D = ST(0) × A. . . . 223
FDIV/FDIVP/FIDIV — FPU Division D = Dst ÷ Src . . . 224
FDIVR/FDIVRP/FIDIVR — FPU Reverse Division D = Src ÷ Dst . . . 224
FPREM — FPU Partial Remainder . . . 225
FPREM1 — FPU Partial Remainder . . . 225
FRNDINT — FPU Round to Integer . . . 226
Advanced Math Instructions . . . 226
FSQRT — FPU ST(0) Square Root . . . 226
FSCALE — FPU Scale ST(0) = ST(0) << ST(1). . . . 226
F2XM1 — FPU ST(0) = $2^{ST(0)} - 1$ . . . 226
FYL2X — FPU ST(0) = y $\log_2 x$ . . . 227
FYL2XP1 — FPU ST(0) = y $\log_2(x+1)$. . . . 227
FXTRACT — FPU Extract Exponent and Significand . . . 227
Floating-Point Comparison . . . 228
FTST — FPU Test If Zero . . . 231
FCOM/FCOMP/FCOMPP — FPU Unordered CMP FP . . . 231
FUCOM/FUCOMP/FUCOMPP — FPU Unordered CMP FP . . . 232
FCOMI/FCOMIP/FUCOMI/FUCOMIP — FPU A ? B and EFLAGS . . . 233
FICOM/FICOMP — FPU A ? B . . . 233
FCMOVcc — FPU Conditional Move . . . 234
FXAM — FPU Examine . . . 235
FPU BCD (Binary-Coded Decimal) . . . 236
FBLD — FPU (BCD Load) . . . 236
FBSTP — FPU (BCD Save and Pop ST(0)) . . . 236
FPU Trigonometry . . . 237
FPTAN — FPU Partial Tangent. . . . 237
FPATAN — FPU Partial Arctangent . . . 237
FSINCOS — Sine and Cosine . . . 237
Pseudo Vec. . . . 241
Pseudo Vec (x86) . . . 241
AMD-SDK . . . 241
3DSMax-SDK . . . 241

Qualcomm Ex. 1050 - 10/25

FSIN — FPU Sine 242
FCOS — FPU Cosine 242
FSINCOS — FPU Sine and Cosine 242
vmp_SinCos (3DNow!) 244
FPU System Instructions 245
FINIT/FNINIT — FPU Init 245
FCLEX/FNCLEX — FPU Clear Exceptions 245
FFREE — FPU Free FP Register 246
FSAVE/FNSAVE — FPU Save X87 FPU, MMX, SSE, SSE2. 246
FRSTOR — FPU Restore x87 State 246
FXSAVE — FPU Save x87 FPU, MMX, SSE, SSE2, SSE3. 247
FXRSTOR — FPU Restore x87 FPU, MMX, SSE, SSE2, SSE3 . . . 247
FSTENV/FNSTENV — FPU Store x87 Environment 247
FLDENV — FPU Load x87 Environment 248
FSTCW/FNSTCW — FPU Store x87 Control Word. 248
FLDCW — FPU Load x87 Control Word 248
FSTSW/FNSTSW — FPU Store x87 Status Word 249
Validating (Invalid) Floating-Point. 249
Exercises 250

Chapter 9 **Comparison 253**
TEST — Logical Compare $A \wedge B$ 254
Indexed Bit Testing 255
BT — Bit Test 255
BTC — Bit Test and Complement 255
BTR — Bit Test and Reset (Clear) 256
BTS — Bit Test and Set 257
SET*cc* — Set Byte on Condition 258
Comparing Operands and Setting EFLAGS 260
CMP — Compare Two Operands 260
COMISS — Compare Scalar SPFP, Set EFLAGS 261
COMISD — Compare Scalar DPFP, Set EFLAGS. 261
UCOMISS — Unordered Cmp Scalar SPFP, Set EFLAGS 262
UCOMISD — Unordered Cmp Scalar DPFP, Set EFLAGS 262
CMPSB/CMPSW/CMPSD/CMPSQ — Compare String Operands . . 263
CMP — Packed Comparison 264
CMPPS/CMPSS/CMPPD/CMPSD — Floating-Point 265
Packed Compare if Equal to (=). 265
Packed Compare if Greater Than or Equal ($\geq$). 266
Packed Compare if Greater Than (>) 266
Extract Packed Sign Masks. 268
PMOVMSKB — Extract Packed Byte (Sign) Mask 268
MOVMSKPS — Extract Packed SPFP Sign Mask. 269
MOVMSKPD — Extract Packed DPFP Sign Mask 269
SCAS/SCASB/SCASW/SCASD/SCASQ — Scan String. 269
REP SCASx 270
CMOV*cc* — Conditional Move 271

Qualcomm Ex. 1050 - 11/25

CMPXCHG — Compare and Exchange 272
CMPXCHG8B — Compare and Exchange 64 Bits 272
CMPXCHG16B — Compare and Exchange 128 Bits 273
Boolean Operations upon Floating-Point Numbers 273
ANDPS — Logical AND of Packed SPFP D = A ∧ B 274
ANDPD — Logical AND of Packed DPFP 274
Pseudo Vec — (XMM) FABS — FP Absolute A = | A | 275
Pseudo Vec — (3DNow!) FABS — FP Absolute A = | A | 275
ORPS — Logical OR of Packed SPFP D = A ∨ B. 276
ORPD — Logical OR of Packed DPFP. 276
XORPS — Logical XOR of Packed SPFP D = A ⊕ B 276
XORPD — Logical XOR of Packed DPFP. 276
Pseudo Vec — FCHS — FP Change Sign A = – A. 277
Pseudo Vec (XMM) 277
Pseudo Vec — (3DNow!) 277
ANDNPS — Logical ANDC of Packed SPFP D = A ∧ (¬B) 278
ANDNPD — Logical ANDC of Packed DPFP. 278
Min — Minimum 278
Pseudo Vec. 279
N×8-Bit Integer Minimum 279
N×16-Bit Integer Minimum. 280
N×SPFP Minimum 281
1×SPFP Scalar Minimum 281
2×DPFP Minimum. 282
1×DPFP Scalar Minimum 282
Max — Maximum 283
N×8-Bit Integer Maximum 283
N×16-Bit Integer Maximum 284
N×SPFP Maximum 285
1×SPFP Scalar Maximum 285
2×DPFP Maximum 286
1×DPFP Scalar Maximum. 286

***Chapter 10* Branching 287**
Jump Unconditionally 287
JMP — Jump 287
Delta JMP. 288
Protected Mode JMP (NEAR) 288
Protected Mode JMP (FAR) 288
Real Mode JMP (NEAR). 288
Real Mode JMP (FAR) 288
Delta JMP 288
Protected Mode JMP (NEAR). 289
Protected Mode JMP (FAR). 289
Real Mode — NEAR or FAR has Same Opcodes 290
Protected Mode — NEAR or FAR has Same Opcodes 290

Qualcomm Ex. 1050 - 12/25

Real Mode — NEAR or FAR has Same Opcodes 291
Protected Mode — NEAR or FAR has Same Opcodes 291
Jump Conditionally 291
Jcc — Branching 291
Delta JMP 291
Branch Prediction 293
Intel Branch Prediction 293
Static Branch Prediction 295
Back-Branch-Taken 295
Forward-Branch-Not-Taken 295
Branching Hints 295
AMD Branch Prediction 296
Branch Optimization 296
PAUSE — (Spin Loop Hint) 299
I-VU-Q 299
JECXZ/JCXZ — Jump if ECX/CX Is Zero 303
LOOPcc 304
LOOP 305
Pancake Memory LIFO Queue 306
Stack 306
PUSH — Push Value onto Stack 307
POP — Pop Value off Stack 307
PUSHA/PUSHAD — Push All General-Purpose Registers 308
POPA/POPAD — Pop All General-Purpose Registers 309
PUSHFD/PUSHFQ and POPFD/POPFQ 310
ENTER — Allocate Stack Frame for Procedure ARGS 310
LEAVE — Deallocate Stack Frame of Procedure ARGS 310
CALL Procedure (Function) 310
CALL 311
Delta CALL 311
Protected Mode CALL (NEAR) 311
Protected Mode CALL (FAR) 311
Real Mode CALL (FAR) 311
Protected Mode CALL (NEAR) 312
Protected Mode CALL (FAR) 312
Real Mode — NEAR or FAR has Same Opcodes 313
Protected Mode — NEAR or FAR has Same Opcodes 313
Real Mode — NEAR or FAR has Same Opcodes 313
Protected Mode — NEAR or FAR has Same Opcodes 313
RET/RETF — Return 313
Calling Conventions (Stack Argument Methods) 314
C Declaration (__CDECL) 315
Standard Declaration (__STDCALL) 317
Fast Call Declaration (__FASTCALL) 317
Interrupt Handling 318
INT/INTO — Call Interrupt Procedure 318
IRET/IRETD/IRETQ — Interrupt Return 320

Qualcomm Ex. 1050 - 13/25

CLI/STI — Clear (Reset)/Set Interrupt Flag 320

Chapter 11 Branchless 323
Function y=ABS(x) 'Absolute' D = | A | 324
Function y=MIN(p, q) 'Minimum' 325
Function y=MAX(p, q) 'Maximum' 326
Graphics 101 — Quick 2D Distance 328

Chapter 12 Floating-Point Vector Addition and Subtraction 333
Floating-Point Vector Addition and Subtraction 334
Vector Floating-Point Addition 334
Vector Floating-Point Addition with Scalar 335
Vector Floating-Point Subtraction 336
Vector Floating-Point Subtraction with Scalar 337
Vector Floating-Point Reverse Subtraction 338
Pseudo Vec 339
Single-Precision Float Addition 339
Single-Precision Float Subtraction 339
Single-Precision Vector Float Addition 340
Single-Precision Vector Float Subtraction 340
Single-Precision Quad Vector Float Addition 340
Single-Precision Quad Vector Float Subtraction 341
Pseudo Vec (x86) 341
vmp_VecAdd (3DNow!) 341
vmp_VecSub (3DNow!) 341
vmp_QVecAdd (3DNow!) 342
vmp_VecAdd (SSE) Unaligned 342
vmp_VecAdd (SSE) Aligned 343
vmp_QVecAdd (SSE) Aligned 343
Vector Scalar Addition and Subtraction 344
Single-Precision Quad Vector Float Scalar Addition 344
Single-Precision Quad Vector Float Scalar Subtraction 344
Special — FP Vector Addition and Subtraction 345
Vector Floating-Point Addition and Subtraction 346
HADDPS/HADDPD/PFACC — Vector Floating-Point Horizontal Addition 346
HSUBPS/HSUBPD/PFNACC — Vector Floating-Point Horizontal Subtraction 347
PFPNACC — Vector Floating-Point Horizontal Add/Sub 348
Exercises 348

Chapter 13 FP Vector Multiplication and Division 351
Floating-Point Multiplication 352
Vector Floating-Point Multiplication 352
(Semi-Vector) DPFP Multiplication 352
SPFP Scalar Multiplication 353
DPFP Scalar Multiplication 353

Qualcomm Ex. 1050 - 14/25

Vector (Float) Multiplication — ADD 354
Pseudo Vec. 354
Single-Precision Float Multiplication 354
Single-Precision Vector Float Multiplication 354
Single-Precision Quad Vector Float Multiplication 354
Single-Precision Quad Vector Float Multiplication-Add 355
Pseudo Vec (x86) 355
vmp_VecMul (3DNow!) 355
vmp_QVecMul (3DNow!) 356
vmp_QVecMAdd (3DNow!) 356
vmp_VecMul (SSE) 356
vmp_VecMul (SSE) Aligned 357
vmp_QVecMul (SSE) Aligned 357
vmp_QVecMAdd (SSE) Aligned 357
Vector Scalar Multiplication 357
Pseudo Vec. 358
Single-Precision Vector Float Multiplication with Scalar 358
Single-Precision Quad Vector Float Multiplication with Scalar. . . 358
Pseudo Vec (x86) 358
vmp_VecScale (3DNow!) 358
vmp_VecScale (SSE) Aligned 359
vmp_QVecScale (SSE) Aligned 359
I-VU-Q. 359
Graphics 101 — Dot Product 359
Pseudo Vec. 360
Single-Precision Dot Product 360
Pseudo Vec (x86) 362
vmp_DotProduct (3DNow!) 362
vmp_DotProduct (SSE) Aligned 363
Graphics 101 — Cross Product 363
vmp_CrossProduct (3DNow!) 364
vmp_CrossProduct (SSE) Aligned 365
Vector Floating-Point Division 366
(Vector) SPFP Division 367
(Semi-Vector) DPFP Division. 367
SPFP Scalar Division 367
DPFP Scalar Division 368
N×SPFP Reciprocal 368
1×SPFP Reciprocal (14-Bit). 369
vmp_FDiv (3DNow!) Fast Float Division 14-Bit Precision. 369
SPFP Reciprocal (2 Stage) (24-Bit). 370
vmp_FDiv (3DNow!) Standard Float Division 24-Bit Precision . . 370
vmp_FDiv (SSE) Standard Float Division 24-Bit Precision. 371
Pseudo Vec. 371
Single-Precision Vector Float Scalar Division. 371
Pseudo Vec (x86) 371

Qualcomm Ex. 1050 - 15/25

vmp_QVecDiv (3DNow!) Fast Quad Float Division 14-Bit Precision . . . 372
vmp_QVecDiv (3DNow!) Standard Quad Float Division 24-Bit Precision . . . 372
vmp_QVecDiv (SSE) Standard Quad Float Division 24-Bit Precision . . . 373
Exercises . . . 373

Chapter 14 Floating-Point Deux . . . 375
SQRT — Square Root . . . 375
1×SPFP Scalar Square Root . . . 377
4×SPFP Square Root . . . 377
1×DPFP Scalar Square Root . . . 378
2×DPFP Square Root . . . 378
1×SPFP Scalar Reciprocal Square Root (15-Bit) . . . 379
Pseudo Vec . . . 379
(Float) Square Root . . . 379
Pseudo Vec (x86) . . . 379
vmp_FSqrt (3DNow!) Fast Float 15-Bit Precision . . . 379
SPFP Square Root (2 Stage) (24-Bit) . . . 380
vmp_FSqrt (3DNow!) Standard Float 24-Bit Precision . . . 380
vmp_FSqrt (SSE) Float Sqrt 24-Bit Precision . . . 381
Vector Square Root . . . 381
Pseudo Vec . . . 381
Vector Square Root . . . 381
Quad Vector Square Root . . . 382
Pseudo Vec (x86) . . . 382
vmp_QVecSqrt (3DNow!) Fast Quad Float SQRT 15-Bit Precision . . . 382
vmp_QVecSqrt (3DNow!) Quad Float Sqrt 24-Bit Precision . . . 383
vmp_QVecSqrt (SSE) Float Sqrt 24-Bit Precision . . . 383
vmp_QVecSqrtFast (SSE) Float Sqrt Approximate . . . 383
Graphics 101 — Vector Magnitude (aka 3D Pythagorean Theorem) . . . 384
2D Distance . . . 384
3D Distance . . . 385
Pseudo Vec . . . 387
Pseudo Vec (x86) . . . 387
vmp_VecMagnitude (3DNow!) . . . 387
vmp_VecMagnitude (SSE) Aligned . . . 388
Vector Normalize . . . 388
Pseudo Vec . . . 388
Pseudo Vec (x86) . . . 389
vmp_VecNormalize (3DNow!) . . . 389
vmp_VecNormalize (SSE) Aligned . . . 390

Chapter 15 Binary-Coded Decimal (BCD) . . . 391
BCD . . . 391
DAA — Decimal Adjust AL (After) Addition . . . 392

Qualcomm Ex. 1050 - 16/25

DAS — Decimal Adjust AL (After) Subtraction 393
AAA — ASCII Adjust (After) Addition 394
AAS — ASCII Adjust AL (After) Subtraction 394
AAM — ASCII Adjust AX (After) Multiplication 395
AAD — ASCII Adjust AX (Before) Division 396
FBLD — FPU (BCD Load) 396
Graphics 101 397
ASCII String to Double-Precision Float 398
ASCII to Double 399

Chapter 16 What CPUID? 403
CPUID 404
Standard CPUID EDX-Feature Flags 408
Intel — Standard CPUID ECX-Feature Flags 409
Intel — Extended #1 CPUID EDX-Feature Flags 410
AMD — Extended #1 CPUID EDX-Feature Flags 410
PIII Serial License 411
Sample CPU Detection Code 411
x86 CPU Detect — Bit Flags 413
x86 CPU Detect — Vendors 413
Cpu Detect — Information 413

Chapter 17 PC I/O 417
IN — Input from Port 418
Vertical Sync 418
OUT — Output to Port 419
INSx — Input from Port to String 420
OUTSx — Output String to Port 420
Serial/Parallel Port for IBM PC 421
Parallel Port 422
Parallel Port Dip Switches 422
Serial Port 425

Chapter 18 System 431
System "Lite" 431
System Timing Instructions 432
RDPMC — Read Performance — Monitoring Counters 432
RDTSC — Read Time-Stamp Counter 432
Calculating Processor Speed 433
80x86 Architecture 435
CPU Status Registers (32-Bit EFLAGS/64-Bit RFLAGS) 436
Protection Rings 437
Control Registers 438
(TPR) Task Priority Registers — (CR8) 439
Debug Registers 439
Cache Manipulation 440
Cache Sizes 440
Cache Line Sizes 441

Qualcomm Ex. 1050 - 17/25

PREFETCH$_x$ — Prefetch Data into Caches 441
LFENCE — Load Fence 442
SFENCE — Store Fence 443
MFENCE — Memory Fence 443
CLFLUSH — Flush Cache Line 443
INVD — Invalidate Cache (WO/Writeback) 444
WBINVD — Write Back and Invalidate Cache 444
System Instructions 444
ARPL — Adjust Requested Privilege Level 445
BOUND — Check Array Index For Bounding Error 445
CLTS — Clear Task Switch Flag 445
HLT — Halt Processor 446
UD2 — Undefined Instruction 446
INVLPG — Invalidate TLB 446
LAR — Load Access Rights 447
LOCK — Assert Lock # Signal Prefix 447
LSL — Load Segment Limit 447
MOV — Move To/From Control Registers 448
MOV — Move To/From Debug Registers 448
STMXCSR — Save MXCSR Register State 448
LDMXCSR — Load MXCSR Register State 449
SGDT/SIDT — Save Global/Interrupt Descriptor Table 449
LGDT/LIDT — Load Global/Interrupt Descriptor Table 449
SLDT — Save Local Descriptor Table 450
LLDT — Load Local Descriptor Table 450
SMSW — Save Machine Status Word 450
LMSW — Load Machine Status Word 451
STR — Save Task Register 451
LTR — Load Task Register 451
RDMSR — Read from Model Specific Register 452
WRMSR — Write to Model Specific Register 452
SWAPGS — Swap GS Base Register 452
SYSCALL — 64-Bit Fast System Call 453
SYSRET — Fast Return from 64-Bit Fast System Call 453
SYSENTER — Fast System Call 453
SYSEXIT — Fast Return from Fast System Call 453
RSM — Resume from System Management Mode 454
VERR/VERW — Verify Segment for Reading 454
LDS/LES/LFS/LGS/LSS — Load Far Pointer 455
Hyperthreading Instructions 457
MONITOR — Monitor 457
MWAIT — Wait 458

Chapter 19 Gfx 'R' Asm 459
Setting Memory 460
Copying Memory 466
Speed Freak 471
Graphics 101 — Frame Buffer 472

Qualcomm Ex. 1050 - 18/25

Graphics 101 — Blit 473
Copy Blit. 475
Transparent Blit 475
Graphics 101 — Blit (MMX). 476
Graphics Engine — Sprite Layered. 476
Graphics Engine — Sprite Overlay 477
Graphics 101 — Clipping Blit 479

Chapter 20 MASM vs. NASM vs. TASM vs. WASM 481
MASM — Microsoft Macro Assembler 483
REPEAT 489
WHILE. 489
FOR 490
Compiler Intrinsics 490

Chapter 21 Debugging Functions 493
Guidelines of Assembly Development. 493
Visual C++ 494
Tuning and Optimization 497
Exception Handling — AKA: Dang that 1.#QNAN. 498
Exceptions 498
FPU Versus MMX 500
Print Output 500
Float Array Print. 501
Vector Print 502
Quad Vector Print 503
Quaternion Print 503
Matrix Print 503
Memory Dump 504
Test Jigs 505
Matrix Test Fill 505
Matrix Splat 506

Chapter 22 Epilogue 507

Appendix A Data Structure Definitions 509

Appendix B Mnemonics 512

Appendix C Reg/Mem Mapping. 528

Glossary 530

References 536

Index 538

Qualcomm Ex. 1050 - 19/25

This material may be protected by Copyright law (Title 17 U.S. Code)

	Intel
P	Pentium
PII	Pentium II (MMX)
SSE	SSE (Katmai NI)
SSE2	SSE2
SSE3	SSE3 (Prescott NI)
E64T	64-bit Memory

	AMD
K6	K6
3D!	3DNow!
3Mx+	3DNow! and MMX Ext.
A64	AMD64

Mnemonic	P	PII	K6	3D!	3Mx+	SSE	SSE2	A64	SSE3	E64T
INC	✓	✓	✓	✓	✓	✓	✓	32	✓	32
SYSRET			✓	✓	✓			✓		64

✓	Mnemonic supported
32	32-bit support only
64	64-bit support only

Each mnemonic will also have a table showing its organization as well as type of data, bit size of data, and operand method for each processor supported.

	Op *rmDst, rSrc(8/16/32/64)*	Signed	
	Op *rDst, rmSrc(8/16/32/64)*		
MMX	Op *mmxDst, mmxSrc/m64*	Signed	64
3DNow!	Op *mmxDst, mmxSrc/m64*	Signed	64
SSE	Op *xmmDst, xmmSrc/m128*	Single-precision	128
SSE2	Op *xmmDst, xmmSrc/m128*	Single-precision	128

rDst	General-purpose register — {8-, 16-, 32-, 64-bit}
rSrc{8/16/32/64}	
m32, m64, m128	32-, 64-, 128-bit memory access
rmDst	General-purpose register and/or memory
mmxDst, mmxSrc	MMX registers
xmmDst, xmmSrc	XMM registers

Note: When general-purpose registers specify 64 bit, such as rDst, rmSrc{8/16/32/64}, then that is *only* available when running in 64-bit mode. Contrarily, 64-bit data is available for MMX and SSE instruction sets.

Some operators take zero, one, two, or three arguments. Unlike some non-80x86 processors, *sourceB* is the same as the *destination*. The 80x86 uses a D += A methodology. When the SIMD operator is being explained, both arguments will be used, such as "The Op instruction

Qualcomm Ex. 1050 - 20/25

80x86 Registers

The Intel 80x86 processor family has internal storage that is referred to as registers. These have been organized into seven groups: general-purpose data registers, segment registers, status registers, control registers, FPU registers used by the FPU (floating-point unit), MMX registers, and XMM registers. Please note that the FPU and MMX registers share storage area and thus are mutually exclusive (only one of them can be used at a time). Each has its own set of functionality.

The following registers and their ranges are for dealing with the SIMD instruction sets directly. They do not include system registers.

Table 3-2: SIMD instruction with register names and bit widths

SIMD Instruction Set	Regs	IA-32	EM64T	Bits
MMX	mm#	(0...7)	(0...7)	64
3DNow!	mm#	(0...7)	(0...7)	64
3DNow! Extensions (3DNow!+)	mm#	(0...7)	(0...7)	64
3DNow! MMX Extensions (MMX+)	mm#	(0...7)	(0...7)	64
SSE	xmm#	(0...7)	(8...15)	128
SSE2	xmm#	(0...7)	(8...15)	128
SSE3	xmm#	(0...7)	(8...15)	128

General-Purpose Registers

The general-purpose registers are organized into two groups of eight registers: The RAX, RBX, RCX, and RDX general registers each have an 8-, 16-, 32-, and 64-bit form, as well as the index registers RSI and RDI, and the stack pointers RBP and RSP. The second set of eight are new registers R8...R15. The instruction pointer RIP has a 16-, 32-, and 64-bit form depending on which mode the processor is running in.

No 64-bit: There is a special case where the AH, BH, CH, and DH are not accessible when the REX prefix is used. REX is used for all extended 64-bit registers as well as SIL, DIL, BPL, and SPL register access.

Qualcomm Ex. 1050 - 21/25

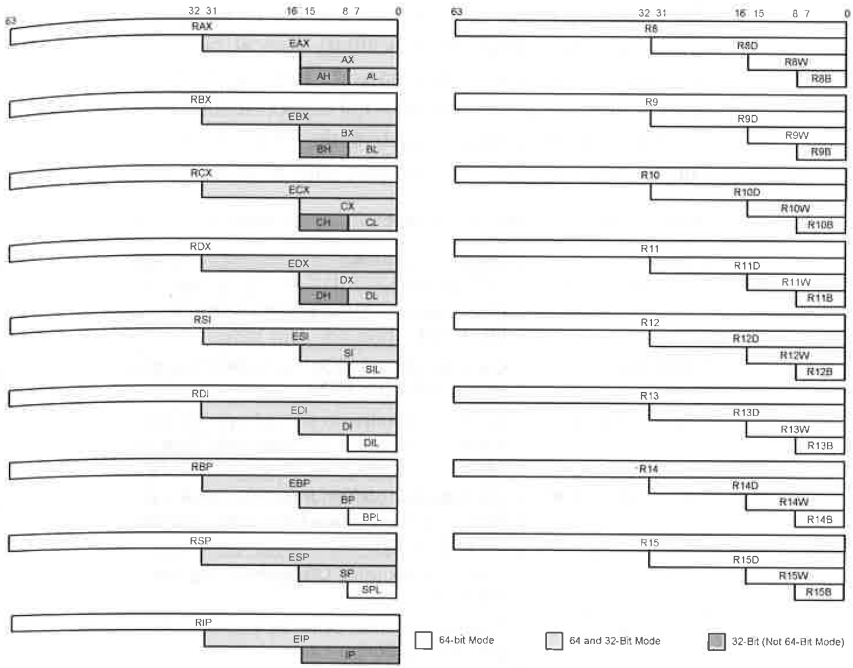


Figure 3-2: 64/32/16-bit general-purpose 80x86 registers

Table 3-3: 64-bit mode registers

64	RAX	RBX	RCX	RDX	RSI	RDI	RBP	RSP	RIP
32	EAX	EBX	ECX	EDX	ESI	EDI	EBP	ESP	
16	AX	BX	CX	DX	SI	DI	BP	SP	
8	AL	BL	CL	DL	SIL	DIL	BPL	SPL	

64	R8	R9	R10	R11	R12	R13	R14	R15
32	R8D	R9D	R10D	R11D	R12D	R13D	R14D	R15D
16	R8W	R9W	R10W	R11W	R12W	R13W	R14W	R15W
8	R8B	R9B	R10B	R11B	R12B	R13B	R14B	R15B

Table 3-4: 32-bit (Protected/Real Mode) registers

32	EAX	EBX	ECX	EDX	ESI	EDI	EBP	ESP	EIP
16	AX	BX	CX	DX	SI	DI	BP	SP	IP
8	AL AH	BL BH	CL CH	DL DH					

Qualcomm Ex. 1050 - 22/25

The general-purpose registers are used as memory pointers, address calculations (displacement w/scaling), logical bit, and mathematical operations. There is an exception as the RSP register is the stack pointer and has limitations such as it does not support scaling and displacement. In relation to the segment registers, the 64-bit stack pointer is used in conjunction with the stack segment-selector (SS) register as a default.

Table 3-5: 64-, 32-, and 16-bit general-purpose registers

Register	Extra Functionality
RAX, EAX, AX	The accumulator. If used as a pointer in Protected Mode uses the DS segment register as a default. DS:[EAX]
RBX, EBX, BX	Used as a data pointer using the DS segment register as a default. DS:[EBX]
RCX, ECX, CX	Used as a counter in string (rep) and loop operations. If used as a pointer in Protected Mode uses the DS segment register as a default. DS:[ECX]
RDX, EDX, DX	Input/output port address. If used as a pointer in Protected Mode uses the DS segment register as a default. DS:[EDX]
RSI, ESI, SI	Source index using DS segment register as a default. DS:[ESI]
RDI, EDI, DI	Destination index using ES segment register as a default. ES:[EDI]
RBP, EBP, BP	Pointer to data on the stack (very similar to ESP); uses the SS segment register as a default. SS:[EBP]
RSP, ESP, SP	Stack pointer used the SS segment register as a default. SS:[ESP]
RIP, EIP, IP	Instruction pointer. CS:[EIP]

Note: The "R" prefix of these registers only became available with the 64-bit versions of the processor. The "E" prefix of these registers became available with the introduction of the 32-bit 386 processor. Prior to that, only 16-bit registers were supported.

Regardless of which mode you were in, you could access either 32-bit or 16-bit registers. But this was with the introduction of a pre-op code. If in 32 bit, you accessed 32-bit registers. In 16 bit, you accessed 16-bit registers. But if you needed to access the alternate type, then a hidden leading prefix was embedded in the binary output:

(**66h**) operand-size prefix
(**67h**) address-size prefix

So from 32-bit code…

```
    mov eax,3
66h mov ax,3
```

Qualcomm Ex. 1050 - 23/25

Logical Packed XOR D=(A⊕B)

Listing 4-8: ...\chap04\pbool\PBool.cpp

```
void vmp_pxor(void *pvD, void *pvA, void *pvB)
{
  uint32 *pD, *pA, *pB;

  pD=(uint32*) pvD;
  pA=(uint32*) pvA;
  pB=(uint32*) pvB;
  *(pD+0) = *(pA+0) ^ *(pB+0);   // {31...0}
  *(pD+1) = *(pA+1) ^ *(pB+1);   // {63...32}
  *(pD+2) = *(pA+2) ^ *(pB+2);   // {95...64}
  *(pD+3) = *(pA+3) ^ *(pB+3);   // {127...96}
}
```

See the code snippets from the previously discussed instruction AND, then substitute the instruction {^, *xor*, *pxor*} for the {&, *and*, *pand*} accordingly.

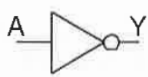
Another use for this operation is as a Boolean NOT (one's complement) operator. By using an input *A* and setting *B* permanently to a logical one, an inverse bit is achieved. Note the following table where zero becomes one and one becomes zero.

A ⊕ B	Y
0 1	1
1 1	0

Figure 4-1: Example of using a logical XOR with a logical true bit to achieve the equivalent of a logical NOT condition.

As can be seen, input A and output Y are the exact opposite. Keep in mind that Boolean logic is bit based and bit isolated so that adjacent bits do not affect each other.

NOT — One's Complement Negation



A	Y
0	1
1	0

Mnemonic	P	PII	K6	3D!	3Mx+	SSE	SSE2	A64	SSE3	E64T
NOT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Qualcomm Ex. 1050 - 24/25

not *rmDst(8/16/32/64)* [Un]signed

This instruction logically inverts the bits of the destination. All ones become zeros, and all zeros become ones.

Flags	O.flow	Sign	Zero	Aux	Parity	Carry
	-	-	-	-	-	-

Flags: None are affected by this opcode.

```
        mov     eax,00000a5a5h
        not     eax
; NOT 00000000000000001010010110100101b   00000a5a5h
;    11111111111111101011010010110101b   0ffff5a5ah
```

And guess what you get when you flip it again?

```
; NOT 11111111111111101011010010110101b   0ffff5a5ah
;    00000000000000001010010110100101b   00000a5a5h
```

The original value! Flip-flop.

NEG — Two's Complement Negation

Mnemonic	P	PII	K6	3D!	3Mx+	SSE	SSE2	A64	SSE3	E64T
NEG	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

neg *rmDst(8/16/32/64)* [Un]signed

This instruction logically inverts the sign of the number.

Flags	O.flow	Sign	Zero	Aux	Parity	Carry
	X	X	X	X	X	X

Flags: Carry is cleared if the destination is 0; else it is set. The other bits are set accordingly from its results.

```
        mov     eax,00000a5a5h
        neg     eax
;   00000000000000001010010110100101b   00000a5a5h (42405)
;   11111111111111101011010010110101b   0ffff5a5bh (-42405)
```

Hint: For a NEG (two's complement negation), use an XOR followed by an increment.

```
    00000000000000001010010110100101b   00000a5a5h (42405)
NOT 11111111111111101011010010110101b   0ffff5a5ah (-42406)
INC 11111111111111101011010010110110b   0ffff5a5bh (-42405)
```

Qualcomm Ex. 1050 - 25/25