X86/WIN32 REVERSE ENGINEERING CHEAT-SHEET

Registers

GENERAL PURPOSE 32-BIT REGISTERS

EAX	Contains the return value of a function call.
ECX	Used as a loop counter. "this" pointer in C++.
EBX	General Purpose
EDX	General Purpose
ESI	Source index pointer
EDI	Destination index pointer
ESP	Stack pointer
EBP	Stack base pointer

SEGMENT REGISTERS

CS	Code segment
SS	Stack segment
DS	Data segment
ES	Extra data segment
FS	Points to Thread Information Block (TIB)
GS	Extra data segment

MISC. REGISTERS

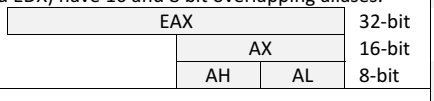
EIP	Instruction pointer
EFLAGS	Processor status flags.

STATUS FLAGS

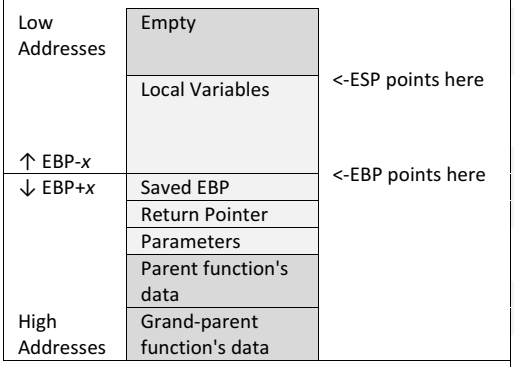
ZF	Zero: Operation resulted in Zero
CF	Carry: source > destination in subtract
SF	Sign: Operation resulted in a negative #
OF	Overflow: result too large for destination

16-BIT AND 8-BIT REGISTERS

The four primary general purpose registers (EAX, EBX, ECX and EDX) have 16 and 8 bit overlapping aliases.



The Stack



Instructions

ADD *<dest>, <source>*	Adds *<source>* to *<dest>*. *<dest>* may be a register or memory. *<source>* may Be a register, memory or immediate value.
CALL *<loc>*	Call a function and return to the next instruction when finished. *<proc>* may be a relative offset from the current location, a register or memory addr.
CMP *<dest>, <source>*	Compare *<source>* with *<dest>*. Similar to SUB instruction but does not Modify the *<dest>* operand with the result of the subtraction.
DEC *<dest>*	Subtract 1 from <dest>. <dest> may be a register or memory.
DIV *<divisor>*	Divide the EDX:EAX registers (64-bit combo) by *<divisor>*. *<divisor>* may be a register or memory.
INC *<dest>*	Add 1 to *<dest>*. *<dest>* may be a register or memory.
JE *<loc>*	Jump if Equal (ZF=1) to *<loc>*.
JG *<loc>*	Jump if Greater (ZF=0 and SF=OF) to *<loc>*.
JGE *<loc>*	Jump if Greater or Equal (SF=OF) to *<loc>*.
JLE *<loc>*	Jump is Less or Equal (SF<>OF) to *<loc>*.
JMP *<loc>*	Jump to *<loc>*. Unconditional.
JNE *<loc>*	Jump if Not Equal (ZF=0) to *<loc>*.
JNZ *<loc>*	Jump if Not Zero (ZF=0) to *<loc>*.
JZ *<loc>*	Jump if Zero (ZF=1) to *<loc>*.
LEA *<dest>, <source>*	Load Effective Address. Gets a pointer to the memory expression *<source>* and stores it in *<dest>*.
MOV *<dest>, <source>*	Move data from *<source>* to *<dest>*. *<source>* may be an immediate value, register, or a memory address. Dest may be either a memory address or a register. Both *<source>* and *<dest>* may not be memory addresses.
MUL *<source>*	Multiply the EDX:EAX registers (64-bit combo) by *<source>*. *<source>* may be a register or memory.
POP *<dest>*	Take a 32-bit value from the stack and store it in *<dest>*. ESP is incremented by 4. *<dest>* may be a register, including segment registers, or memory.
PUSH *<value>*	Adds a 32-bit value to the top of the stack. Decrements ESP by 4. *<value>* may be a register, segment register, memory or immediate value.
ROL *<dest>, <count>*	Bitwise Rotate Left the value in *<dest>* by *<count>* bits. *<dest>* may be a register or memory address. *<count>* may be immediate or CL register.
ROR *<dest>, <count>*	Bitwise Rotate Right the value in *<dest>* by *<count>* bits. *<dest>* may be a register or memory address. *<count>* may be immediate or CL register.
SHL *<dest>, <count>*	Bitwise Shift Left the value in *<dest>* by *<count>* bits. Zero bits added to the least significant bits. *<dest>* may be reg. or mem. *<count>* is imm. or CL.
SHR *<dest>, <count>*	Bitwise Shift Left the value in *<dest>* by *<count>* bits. Zero bits added to the least significant bits. *<dest>* may be reg. or mem. *<count>* is imm. or CL.
SUB *<dest>, <source>*	Subtract *<source>* from *<dest>*. *<source>* may be immediate, memory or a register. *<dest>* may be memory or a register. (source = dest)->ZF=1, (source > dest)->CF=1, (source < dest)->CF=0 and ZF=0
TEST *<dest>, <source>*	Performs a logical OR operation but does not modify the value in the *<dest>* operand. (source = dest)->ZF=1, (source <> dest)->ZF=0.
XCHG *<dest, <source>*	Exchange the contents of *<source>* and *<dest>*. Operands may be register or memory. Both operands may not be memory.
XOR *<dest>, <source>*	Bitwise XOR the value in *<source>* with the value in *<dest>*, storing the result in *<dest>*. *<dest>* may be reg or mem and *<source>* may be reg, mem or imm.

Assembly Language

Instruction listings contain at least a mnemonic, which is the operation to be performed. Many instructions will take operands. Instructions with multiple operands list the destination operand first and the source operand second *(<dest>*, <source>). Assembler directives may also be listed which appear similar to instructions.

ASSEMBLER DIRECTIVES

DB *<byte>*	Define Byte. Reserves an explicit byte of memory at the current location. Initialized to <byte> value.
DW *<word>*	Define Word. 2-Bytes
DD *<dword>*	Define DWord. 4-Bytes

OPERAND TYPES

Immediate	A numeric operand, hard coded
Register	A general purpose register
Memory	Memory address w/ brackets [ ]

Terminology and Formulas

Pointer to Raw Data	Offset of section data within the executable file.
Size of Raw Data	Amount of section data within the executable file.
RVA	Relative Virtual Address. Memory offset from the beginning of the executable.
Virtual Address (VA)	Absolute Memory Address (RVA + Base). The PE Header fields named VirtualAddress actually contain Relative Virtual Addresses.
Virtual Size	Amount of section data in memory.
Base Address	Offset in memory that the executable module is loaded.
ImageBase	Base Address requested in the PE header of a module.
Module	An PE formatted file loaded into memory. Typically EXE or DLL.
Pointer	A memory address
Entry Point	The address of the first instruction to be executed when the module is loaded.
Import	DLL functions required for use by an executable module.
Export	Functions provided by a DLL which may be Imported by another module.
RVA->Raw Conversion	Raw = (RVA - SectionStartRVA) + (SectionStartRVA - SectionStartPtrToRaw)
RVA->VA Conversion	VA = RVA + BaseAddress
VA->RVA Conversion	RVA = VA - BaseAddress
Raw->VA Conversion	VA = (Raw - SectionStartPtrToRaw) + (SectionStartRVA + ImageBase)