MASM Reference
Ethan Rietz

assemble → link → load → run
fetch → decode → execute
each memory address identifies a single byte or 8 bits

Numbers

Dec	Bin	Hex	Dec	Bin	Hex
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

If first number of hex is 0-7, number is positive. If 8-F, then negative.

Floats

Single (real4) 32 bits: exponent: 8, mantisa: 23
Double (real8) 64 bits: exponent: 11, mantisa: 52
Extend (real10) 80 bits: exponent: 15, mantisa: 64

Memory Units

Type	Name	bits	Range
Byte	BYTE	8	[0 , 2^{8}-1]
Signed Byte	SBYTE	8	[-2^{7} , 2^{7}-1]
Word	WORD	16	[0 , 2^{16}-1]
Signed Word	SWORD	16	[-2^{15} , 2^{15}-1]
Doubleword	DWORD	32	[0 , 2^{32}-1]
Signed Doubleword	SDWORD	32	[-2^{31} , 2^{31}-1]
Quadword	QWORD	64	[0 , 2^{64}-1]
Signed Quadword	SQWORD	64	[-2^{63} , 2^{63}-1]
Double Quadword	OWORD	128	[0 , 2^{128}-1]
Signed Double Quadword	SOWORD	128	[-2^{127}, 2^{127}-1]

Status Flags

Status flags

EFLAGS register controls or reports the status of the processor. Each bit is its own flag described below.

CF or CY Carry flag: unsigned arithmetic too big
PF or PE Parity flag: set if LSB contains even number of ones
AF or AC Auxiliary flag: binary coded decimal arithmetic
ZF or ZR Zero flag: set if result is zero
SF or PL Sign flag: set equal to the MSB of signed integer
OF or OV Overflow flag: signed integer arithmetic

Control Flags

DF Direction flag: direction of memory traversal for strings

Segment registers

CS Points to beginning of code segment
DS Points to beginning of data segment
ES FS GS Point to other data storage segments
SS Points to stack segment

Instruction Pointer

EIP contains the OFFSET (in code segment) of next instruction to be executed. Adding EIP to CS gives real address of next instruction.

Registers

EBX the only real general purpose register
EAX used mainly for ALU operations
EDX extension of EAX for operations beyond 32 bits
ECX used as a counter (e.g. LOOP and REP)
ESI/EDI for memory transfer functions (source and destination)
ESP Stack pointer (points to top of runtime stack)
EBP points to the base of a stack frame

Can access EAX, EBX, ECX, and EDX sub-registers as AX, BX, CX, and DX. The high and low addresses are further named AH, AL, etc...

Instructions

INSTR op1, op2
MOV copy data from op2 to op1
MOVSX moves a signed value into a register and sign-extends it
MOVZX moves an unsigned value into a register and zero-extends it
XCHG exchange the contents of operands (cant do mem to mem)

ADD add op2 to op1, store result in op1
SUB sub op2 from op1, store result in op1
INC add 1 to op1, store result in op1 (affects carry flag)
DEC subtract 1 from op1, store result in op1 (affects carry flag)

MUL multiply val in EAX by op2, store result in EDX:EAX
IMUL signed multiply val in EAX by op2, store result in EDX:EAX
DIV div val in EAX by op2. Quotient stored in EAX, remainder stored in EDX. Pre: EDX must be set equal to 0.
IDIV signed division similar to DIV. Remainder has same sign as dividend.

CDQ Converts signed DWORD in EAX to a signed quad word in EDX:EAX by extending the high order bit of EAX throughout EDX
CWD Converts a signed word in AX to a signed doubleword in EAX by extending the sign bit of AX throughout EAX.
CBW Converts byte in AL to word Value in AX by extending sign of AL throughout register AH.

AND performs a logical AND of op1 and op2 replacing op1 with result. Modifies flags.
OR logical inclusive OR of op1 and op2. Result stored in op1. Modifies flags.
XOR bitwise exclusive OR of op1 and op2. Result stored in op1. Modifies flags.

CMP subtracts op2 from op1 and updates flags. Results not saved.
LOOP decrements CX by 1 and transfers control to

"label:" if CX is not zero. The "label" operand must be within -128 or 127 bytes of the instruction following the loop instruction

Jcond	flags	Jcond	flags
JE	ZF=1	JE	ZF=1
JNE	ZF=0	JNE	ZF=0
JG	SF=0 and ZF=0	JA	CF=0 and ZF=0
JGE	SF=OF	JAE	CF=0
JNGE	ZF=1 or SF!=OF	JNAE	CF=1
JL	SF!=OF	JB	CF=1
JLE	SF!=OF	JBE	CF=1 or ZF=1
JNL	ZF=1 or SF!=OF	JNB	CF=0
JNLE	SF=0 and ZF=0	JNBE	CF=0 and ZF=0

JZ jump if result is zero
JNZ jump if result is non-zero
JS jump if sign flag set
JNS jump if sign flag not set
JO jump if overflow flag set
JNO jump if overflow flag not set
JC jump if carry flag set
JNC jump if carry flag not set
JP jump if parity flag set
JNP jump if parity flag not set

JCXZ jump if counter register CX is zero
JECXZ jump if counter register ECX is zero

= Declares a (integer literal) constant.
EQU Equate a constant equal to text or expression.
TEXTEQU Creates a text macro.

PUSH pushes operand onto runtime stack and decrements ESP
POP pops top of stack into operand and increments ESP
PUSHAD push all 32 bit registers onto stack
POPAD pop all 32 bit registers off of stack
PUSHA push all 16 bit registers onto stack
POPA pop all 16 bit registers off of stack
PUSHFD pushes the EFLAGS register onto the stack
POPFD pops top of stack off into EFLAGS

CALL push current value of EIP onto stack and jump to procedure. Decrements ESP by 4.
RET pops the top of the stack into EIP, the instruction pointer.

Irvine32

ReadInt reads a signed integer into EAX
ReadDec reads an unsigned integer into EAX
ReadString reads a string into EAX. Pre: mem offset in EDX and size of mem destination in ECX.
ReadChar
WriteInt writes signed int to console. Pre: value in EAX
WriteDec writes unsigned int to console. Pre: value in EAX
WriteString writes string to console. Pre: mem offset in EDX
WriteChar writes single character to stdout.
CrLf Carriage return line feed

TYPE Number of bytes in the data type used in declaration
OFFSET returns address offset from start of data segment of data label
LENGTHOF Length used in declaration (aka number of elements in array)
SIZEOF Size of memory assigned in declaration. (same as LENGTHOF × TYPE)

STD Set direction flag. Primitives decrement by size (in bytes) of the TYPE. Used to move backwards through array.
CLD Clear direction flag. Primitives increment by size (in bytes) of the TYPE. Used to move "forward" through an array.

LOD(SB)(SW)(SD) Load mem addressed by ESI into accumulator
STO(SB)(SW)(SD) Store accumulator contents into memory addressed by EDI
MOV(SB)(SW)(SD) "Move" copy data from mem addressed by ESI into memory addressed by EDI
CMP(SB)(SW)(SD) Compare contents of two mem locations addressed by ESI and EDI
SCA(SB)(SW)(SD) "Scan" compare accumulator to memory addressed by EDI
SB, SW, SD In above instructions refer to BYTE, WORD, and DWORD sized instructions.

REP Repeat string primitive and decrement ECX while ECX > 0
REPZ Same as REP but while Zero flag is set and ECX > 0
REPE Same as REPZ. Repeat while equal.
REPNZ Same as REP but while Zero flag is clear and ECX > 0
REPNE Same as REPNZ. Repeat while not equal.

LOCAL Creates local variable. Creates stack frame, terminates stack frame, makes space on stack for local variables, provides labels to reference stack locations.
REQ Marks macro argument as required.

FINIT Must be executed before any other FPU instructions
FLD Loads floating-point value on FPU stack
FILD Loads like FLD and converts int to REAL10
FST Stores floating-point value from ST(0) into mem location
FIST Like FST but stores as int in memory
FSTP Stores like FST but also pops off ST(0) from stack.

FADD Add source to destination, overwrite destination
FSUB Subtract source from destination
FMUL Multiply source by destination
FDIV Divide destination by source
Note: all above operations occur with old ST(0) and ST(1) and result is stored in new ST(0).

FCHS Invert sign of ST(0). No operands.
FABS Clear sign of ST(0). No operands.

Extra

Big MSB stored first (lower) in memory. LSB stored last (higher).
Little LSB stored first (lower) in memory. MSB stored last (higher).
x86-64 systems are little endian.

$$T_{parallel} = fT + \frac{(1-f)T}{n}$$