|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **REG Name** | **Type** | **Special Uses** | **Comments** | Largest 32b |
| EAX, AX, AH, AL | General | Result of add, dividend in DIV, and quotient= AL, Remainder= AH | MUL:  Mutiplicand, AL, AX, EAX  Product Etx= AX, DX:AX, EDX:EAX | Signed Int  (2^31) - 1 |
| EBX, BX, BH, BL | General | None |  | $-newarray |
| ECX, CX, CH, CL | General | Loop counter |  | Div by type |
| EDX, DX, DH, DL | General | DIV Remainder =DX or EDX | MUL: Product extension = EDX | Size = #elm. |
| EBP, BP | Multi | Extended Frame Pointer | **STATUS FLAGS** | **Instruction=** |
| ESP, SP | Multi | Extended Stack | Carry (CF)- unsigned is too large | Label, Mn, |
| ESI, SI | Multi | Ext Source index | Overflow (OF)- signed is too large | Operand, |
| EDI, DI | Multi | Ext Destination index | Sign (SF)- result = neg | comment |
| EFLAGS | Special | Can’t be directly accessed | Zero(ZF)- result = 0 |  |
| EIP(instruction pointer) | Special | Can’t be directly accessed | Auxiliary Carry (AC)- from bit 3-4 |  |
| CS, ES, SS, FS, DS, GS | Segment (16big) |  | Parity (PF)- LSB = even 1 bits |  |
| (unsigned) JA = lo>ro, JAE | JB = lo<ro, JBE | (Signed JG) = lo > ro, JGE | JL = lo < ro, JLE ;;;;;;; **JE = lo = ro** | **JNE= lo !=ro** |
| **CMP** implied subtraction | D < S CF-1 | D > S CF-0; D=S ZF-1 | (signed) D<S SF!=OF; D>S SF=OF | D=S ZF-1 |

XCHG-REG/REG, MEM/REG, REG/MEM *NO!!! MEM/MEM*

MOVSX- extended sign for sized, MOVZX- extend sign for unsigned

ESP- Modified by PUSH, POP, CALL, RET(PUSH dec esp by 4 POP inc 4, CALL dec 4, RET inc 4)

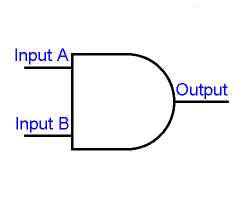
WRITEDEC-unsigned (From EAX

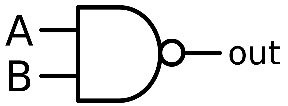
RandomRange- produces random number

PROC USES registers (pushes named registers then pops at the end (changes esp before ebp is set so changes addressing)

READSTRING (pre- OFFSET IN EAX)

WriteInt-signed (from EAX)

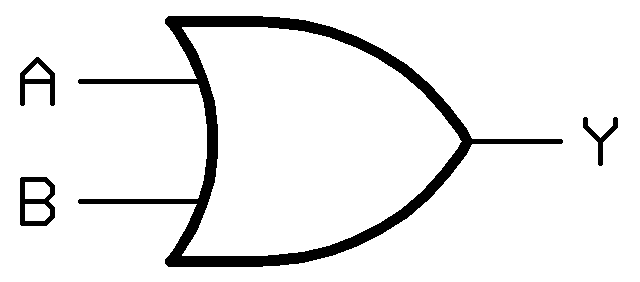
CALL- moves return address to ESP



[This Photo](https://commons.wikimedia.org/wiki/File:Nand-gate-en.svg) by Unknown Author is licensed under [CC BY-SA](https://creativecommons.org/licenses/by-sa/3.0/)

AND – 1+1 = 1; use it to clear bits where ever the zeros are, NAND = 1+1=0

OR 1+0=1 0+1=1 1+1 = 1 use it to set bits use 1 to set NOR = 0+0=1



[This Photo](https://commons.wikimedia.org/wiki/File:Logic-gate-or-us.png) by Unknown Author is licensed under [CC BY-SA](https://creativecommons.org/licenses/by-sa/3.0/)

XOR 1+1=0, 0+0=0 else =1, XN0R 0+0=1, 1+1=1 (tradition addition) attach a carry gate and you get an half adder



[This Photo](https://ucdavis-ece-prelim.wikispaces.com/Combinational+Logic) by Unknown Author is licensed under [CC BY-SA](https://creativecommons.org/licenses/by-sa/3.0/)

SINGLE FLOAT: S= 1 E=8 Sig= 23

Double Float: s= 1 E=11 Sig = 52

RET – adds 4 to esp (Hds Up for pops after a procedure call when doing RET ? questions) Pops stack into EIP(instruction pointer

|  |  |  |  |
| --- | --- | --- | --- |
| Register Indirect Addressing | Base Index Addressing | Indexed (global) | parrallism algo, n=#proc, f=sequential code T= time for single |
| mov eax, [esi] | mov eax, [edx + ebx] | list[esi] | fT +((1-f)T/n) |

push OFFSET Practice (address will be [ebp +12]

push practice2 (address will be [ebp + 8], then call procedure

infix to postfix algorithm- make binary tree- do post-order transversal (L, R, Root)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **(** | **Operand** | **Operator** | **)** | **pipeling algo: tasktime \* instruction + (n-1\*tasktime)** |
| go left | insert | go up, insert, go right | go up | n= number of stages |

maxtrix address algo: [i] x size then convert to hex, ex. list DWORD 42 DUP(?) 14th element= 13 x 4 = 52d= 34h

maxtrix [i][j] address algo: [i] x number of columns + 1D algo

RandomRange Algo: [0…N-1] in eax, put N(range) in eax

To generate a random number in [lo .. hi]:

•Find number of integers possible in [lo .. hi] : range = hi – lo +1

•Put *range* in **eax**, and call *RandomRange*  •Result in **eax** is in [0 .. *range* -1] •Add *lo* to **eax**.

PTR- overrides the default type of a label. Can access part of a register

Can be used to fuck around with an array and practice little endian BS example: myBytes BYTE 12h, 34h, 56h, 78h

mov ax, WORD PTR mBytes ; AX =3412h (it reverses that chunk)

TYPE returns the size in bytes of a single element, LENGTHOF returns the number of elements in a data declaration

SIZEOF returns LENGTHOF x TYPE

Declare a pointer like this: ptr DWORD list(were list is a previously declared variable

LODSB moves byte at [esi] in AL register, inc esi if cld, dec if std

STOSB moves byte in AL to memory at [edi] in cedi if cld, dec if std

|  |  |
| --- | --- |
| READINT algo: x = x = 10 \* x + (str[i] - 48)  0 mov eax, 0  lodsb ; next byte is now in al  ;make sure the ASCII is a digit  cmp eax, 48  jb error  cmp eax, 57  ja error  ;conversion algo: x = 10 \* x + (str[i] - 48)  ;subtract (str[i] is in eax)  sub eax, 48  ;sav results  mov ebx, eax  ;prep mul goodInt into eax, multiplier into edx  mov eax, [edi]; goodInt  mov edx, 10  ; mul x  mul edx  ;validate size of integer  jo error  ;add results  add eax, ebx  ;sav results in goodInt  mov [edi], eax | make infix, postfix and then push and compute in that order  FLD MemVar = push ST0i) to ST(i+1), load st with memVar  FST- move top to memory, leave result  FSP- pop top of stack  •FADD: Addition (pop top two, add, push result)  •FSUB: Subtraction  •FMUL: Multiplication  •FDIV: Division  •FDIVR: Division (reverses operands)  •FSIN: Sine (uses radians)  •FCOS: Cosine (uses radians)  •FSQRT: Square Root  •FABS: Absolute Value  •FYL2X: Y \* log2 (X) (X is in ST(0), Y is in ST(1)  •FYL2XP1: Y \* log2 (X)+1 |