Assembler Final Cheat Sheet

by joshfedo94 via cheatography.com/37022/cs/11664/

maths

IMUL	Multiply ax by what ever is specified (for 32 bit store in DX:AX)
DIV	(16 bit) takes the operand and divides it by AX and stores it in AL with remainder in AH. for 32 bit it used DX:AX pair and leaves awn is AX and remainder in DX
MUL	Multiply ax by what ever is specified (all unsigned)
CWD	convers the word in AX to a double word in DX:AX

Convert num to bytes

ASCII	1 byte per char
unsigned	2 bytes for 5 chars
bcd	2 bytes for 5 chars

code examples

Write code that would find the sum 6+12+18...+300 and store it in var tot

```
MOV TOT, 0; MOV AX,6; LOOP: ADD TOT, AX; ADD AX, 6 CMP AX,300; JLE LOOP
```

write code that is assembler equiv: if(x<y){x++;}esle{y+= 2}

```
MOV AX, X; CMP AX,Y; JGW ELSE; ADD AX,1 JMP END; ELSE: ASS Y,2; END:
```

move 500 bytes of data TABLE1 to TABLE 2 using MOVSB

code examples (cont)

```
LEA SI, TABLE1; LEA DI, TABLE2; MOV CX, 500; CLD; LPTOP: MOVSB; LOOP LPTOP
```

MOVE 500 WORDS OF DATA FROM TABLE1 TO TABLE 2 USING INDEXING

```
MOV CX,500; MOV BX,0; LPTOP: MOV AX, TABLE1[BX]; MOV TABLE2[BX],AX; ADD BX, 2; LOOP LPTOP
```

count the number of blanks in the 1000 byte string of chars referanced by table 1 using scasb

```
MOV AX, SEG TABLE1; MOV ES, AX; MOV AL, ' '; LEA DI, TABLE1; COV CNT,0; CLD; LPTOP: SC ASB; JNE: SKIP; INC CNT; SKIP: LOOP LPTOP
```

binary

signed magnitude	Very left bit is 0 for + num and 1 for -num
twos compliment	flip the bits and add 1
27 excises	add the num to 128 then convert to binary
ones compliment	flip all bits
unsigned	all bits count but its a positive num

Bit shifting

RCR	rotate right last bite gets stored in carry and carry gets pushed to the first bite
SHL	Shift left into cf
TEST	The TEST operation sets the flags CF and OF to zero. The SF is set to the most significant bit of the result of the AND. If the result is 0, the ZF is set to 1, otherwise set to 0. The parity flag is set to the bitwise XNOR of the least significant byte of the result, 1 if the number of ones in that byte is even, 0 otherwise. The value of AF is undefined.
SAR	shift right into carry but keep the signed bit the same
CMC	invert CF
ROL	roatate left into the last bit and the carry flag
CLC	CF = 0
STC	CF = 1

adressing

tab[di]	indexed adressing [offset + ds *10 + DI]
[bx] [di]	base indexing[reg 1 + reg 2 + ds * 10]
[si]	register indirect[ds*10 + reg1]
[bp]	base addressing[ss*10 + reg1]

Loads

LODSB	loads al wiht copy of DS:SI. IF DF = 0 then si++
LODSW	loads ax wiht copy of DS:SI. IF DF = 0 then si ++
STOSB	replace byte pointed to by ES:DI with a copy of AL and incs DI
STOSW	replace byte pointed to by ES:DI with a copy of AX and incs DI
CLD	clears DF
STD	set DF
MOVSW	replaces byte pointed to by ES:DI with word at DS:SI. Moves SI:DI by 2
MOVSB	copies the byte at [DS:SI] or [DS:ESI] to [ES:DI] or [ES:EDI]. It then increments or decrements (depending on the direction flag: increments if the flag is clear, decrements if it is set) SI and DI (or ESI and EDI).