**Assembly Short Qs & Definitions**

**SR** - Status Register

**IP** - instruction Pointer

**SP** - Stack Pointer

**PC** - Programme Counter

**SR Data Flags**

* Zero flag
* Carry flag
* Sign flag
* Negative flag
* Overflow flag

**JZ** -only jump if the Zero flag is set

**JNZ** - only jump if the Zero flag is not set

**JS** - only jump if the Sign flag is set

**JNS** - only jump if the Sign flag is not set

**JC** - only jump if the Carry flag is set

**JNC** - only jump if the Carry flag is not set

**JO** - only jump if the Overflow flag is set

A **Stack** is a useful way of working with memory to store data into memory.

**Advantages**:

Useful for simple storage and retrieval.

Useful for reversing lists of things.

Stack code is more compact.

**Stack Overflow**: Pushing too much data onto the stack. Overwriting active parts of code.

**Stack Underflow**: Popping too much data from the stack. Reading data from the stack that your program never even put there.

**PUSHF**:push the flags register.

**POPF**: pop the number at the top of the stack into the flags register.

**Procedures** are a useful way of defining a repetitive sequence of tasks.

The **ret instruction** transfers control to the return address located on the stack.

The **call instruction** places the address on the stack.

**SHR** - Bit division by power of 2 (SHR AL, 2 Divides by 2^2)

**SHL** - Bit multiplication by power of 2 (SHL AL,3 Multiplies by 2^3)

**Bit Operators**

**XOR** - Flip every odd bit ( bit s # 1 , 3 , 5 , 7 ) of AL

MOV BL , A A ; A A = 1 0 1 0 1 0 1 0

XOR AL , BL

**OR** - Set every odd bit of AL ; leaving every even bit alone

MOV BL , A A ; AA = 1 0 1 0 1 0 1 0

OR AL , BL

**AND** -Clear every bit of AL (but leaving the first bit alone).

MOV BL,80 ; 80 = 1000 0000

AND AL,BL

**NOT** - flip every bit o f AL

MOV AL , FE ; F F = 1 1 1 1 1 1 1 0

NOT AL ; AL is now 0 1 0 0 0 0 0 0 0 1 (in binary)

An **interrupt** is a signal to the CPU to stop what it is doing, go execute code somewhere else and (when ready) resume what it was doing before it was interrupted.

Interrupt 0x02 (**The Timer**)

Interrupt 0x03 (**The Numeric Keypad**)

Interrupt 0x04 (**The advanced Keyboard**)

**STI** - Sets the interrupt flag to 1

**CLI** - Sets the interrupt flag to 0

**ISR** - Interrupt Service Routine

**IRET** gets the old (saved) value for the IP from the stack and restores it

**Machine code** - is what the CPU understands natively (Binary).

**Assembly** - is a human readable representation of machine code. It also still has such niceties as readable labels instead of addresses. The CPU cannot understand assembly code.

**Three techniques for passing parameters to a procedure:**

* Passing Parameters via the Stack
* Passing Parameters via Memory Locations
* Passing Parameters via Registers

**MNZ** (Move if Not Zero) - MNZ 0, AL ;move al to 0 if it is not already 0