8086 Assembly Language Basics

Registers Overview

General Purpose Registers:

- **AX (Accumulator Register):**
  - Primary register for arithmetic, logic, and data transfer operations.
  - Divided into **AH** (High byte) and **AL** (Low byte).
- **BX (Base Register):**
  - Used to hold memory addresses for addressing data.
  - Divided into **BH** (High byte) and **BL** (Low byte).
- **CX (Count Register):**
  - Primarily used as a loop counter.
  - Divided into **CH** (High byte) and **CL** (Low byte).
- **DX (Data Register):**
  - Used for I/O operations and extended arithmetic operations.
  - Divided into **DH** (High byte) and **DL** (Low byte).

Pointer and Index Registers:

- **SI (Source Index):** Used for source addressing in string operations.
- **DI (Destination Index):** Used for destination addressing in string operations.
- **BP (Base Pointer):** Points to data on the stack.
- **SP (Stack Pointer):** Points to the top of the stack.

Important Concepts

DB

- **Define Byte:** A directive to allocate memory and initialize it with byte-sized data.

```
msg DB "Hello, 8086!", 0Dh, 0Ah, "$"
num DB 100
```

- `msg` stores a string ending with `$` (used by DOS interrupts). - `num` allocates a single byte initialized to 100.

Labels

- A marker in the code that acts like a bookmark for loops or jumps.

```
print_loop:
    ; Code here
```

MOV Instruction

- Transfers data between registers or between memory and registers.

```
MOV AX, BX  ; Copy the value of BX into AX
MOV AL, 5   ; Load 5 into AL (lower byte of AX)
```

Code Example

Complete Program:

```
.model small
.stack 100h
.data
    msg DB "Hello, 8086!", 0Dh, 0Ah, "$"   ; Message to print
.code
main proc
    ; AX: Arithmetic example
    MOV AX, 5        ; Load 5 into AX
    ADD AX, 10       ; Add 10 to AX (AX = 15)

    ; BX: Addressing example
    MOV BX, OFFSET msg  ; Load address of 'msg' into BX

    ; CX: Loop counter
    MOV CX, 5          ; Set loop counter to 5
print_loop:
    MOV AH, 2          ; Function to print a character
    MOV DL, '*'        ; Load '*' into DL
    INT 21h            ; Print the character
    LOOP print_loop    ; Decrement CX and jump if CX > 0

    ; DX: I/O operation
    MOV AH, 9          ; Function to print a string
    MOV DX, BX         ; Load address of 'msg' into DX
    INT 21h            ; Print the string

    HLT                ; Halt execution
main endp
end main
```

Code Explanation:

1. **.model small:** Specifies the memory model (small = single code and data segment).

2. **.stack 100h:** Reserves 256 bytes (100h) for the stack.

3. **.data:** Segment for declaring variables and data.

4. **msg DB Ḧello, 8086!,̈ 0Dh, 0Ah, $̈:̈** Defines a string ending with `$`, used by `INT 21h` to print strings.

5. **MOV AX, 5:** Loads the value 5 into the AX register.

6. **ADD AX, 10:** Adds 10 to the value in AX (AX becomes 15).

7. **MOV BX, OFFSET msg:** Stores the memory address of `msg` into the BX register.

8. **MOV CX, 5:** Initializes the loop counter to 5.

9. **print_loop:** Label marking the start of the loop.

10. **MOV AH, 2:** Prepares for a single-character print operation.

11. **MOV DL, '*':** Loads the ASCII value of * into DL.
12. **INT 21h:** DOS interrupt for performing I/O operations.
13. **LOOP print_loop** Decrements CX and jumps to print_loop if CX ¿ 0.
14. **MOV AH, 9:** Prepares for a string print operation.
15. **MOV DX, BX:** Loads the address of msg (stored in BX) into DX.
16. **HLT:** Halts the program.

Quick Revision (Bullet Notes)

- **AX:** Arithmetic and data transfer.
- **BX:** Memory addressing.
- **CX:** Loop counter.
- **DX:** I/O operations.
- **DB:** Define byte-sized variables or strings.
- **Labels:** Used for loops or jumps.
- **MOV Instruction:** Transfers data between registers/memory.
- **INT 21h:** DOS interrupt for I/O.
  - AH = 2: Print a single character (character in DL).
  - AH = 9: Print a string (address in DX).
- **HLT:** Stops execution.

Topics Covered

Today, we explored more advanced concepts in assembly language programming, focusing on conditional branching, printing values, and structured control flow. Below is a detailed summary of the key topics:

1 Conditional Branching

- In assembly, conditional branching is handled using instructions like `CMP` (compare) and conditional jumps such as:
    - `JE` (Jump if Equal)
    - `JG` (Jump if Greater)
    - `JL` (Jump if Less)
    - `JNE` (Jump if Not Equal)
- Example of a simple conditional branch:

```
CMP AX, BX
JG GREATER
; Code for else block
JMP END_IF
GREATER:
; Code for greater block
END_IF:
```

2 Printing Values

- To print data, we use the `INT 21H` interrupt with specific function codes:
    - `AH = 09H` to display a string. The string must be terminated by a `$` symbol.
    - `AH = 02H` to display a single character (with the character stored in `DL`).
- Example of printing a string:

```
LEA DX, STRING
MOV AH, 09H
INT 21H
```

- Example of printing a single character:

```
MOV DL, 'A'
MOV AH, 02H
INT 21H
```

3 Structured Control Flow

- Assembly does not have built-in blocks for conditional or nested control flow.
- Explicit labels and jumps are used to define the structure of conditions and loops.
- Example of a nested condition:

```
CMP AX, BX
JG GREATER
CMP CX, DX
JE EQUAL
; Else-Else block
JMP END_NESTED
EQUAL:
; Else-If block
JMP END_NESTED
GREATER:
; If block
END_NESTED:
; Code continues
```

4 LEA Instruction

- `LEA` (Load Effective Address) is used to load the address of a variable into a register, typically `DX`.

- Example:

  ```
  LEA DX, STRING
  ```

  This moves the address of STRING into the DX register.

5 Complete Example: Nested Conditions

```
.MODEL SMALL
.STACK 100H

.DATA
    NUM1 DW 7
    NUM2 DW 5
    NUM3 DW 9
    NUM1_MSG DB 'NUM1 is the greatest$', 0
    NUM2_MSG DB 'NUM2 is the greatest$', 0
    NUM3_MSG DB 'NUM3 is the greatest$', 0

.CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX

    ; Compare NUM1 and NUM2
    MOV AX, NUM1
    CMP AX, NUM2
    JG CHECK_NUM3           ; If NUM1 > NUM2, check against NUM3

    ; Else, compare NUM2 and NUM3
    MOV AX, NUM2
    CMP AX, NUM3
    JG NUM2_IS_GREATEST     ; If NUM2 > NUM3, NUM2 is greatest

    ; Else, NUM3 is greatest
    MOV AH, 09H
    LEA DX, NUM3_MSG
    INT 21H
    JMP END_PROGRAM
```

```
NUM2_IS_GREATEST:
    ; Code for NUM2 > NUM3
    MOV AH, 09H
    LEA DX, NUM2_MSG
    INT 21H
    JMP END_PROGRAM

CHECK_NUM3:
    ; Compare NUM1 and NUM3
    CMP AX, NUM3
    JG NUM1_IS_GREATEST     ; If NUM1 > NUM3, NUM1 is greatest

    ; Else, NUM3 is greatest
    MOV AH, 09H
    LEA DX, NUM3_MSG
    INT 21H
    JMP END_PROGRAM

NUM1_IS_GREATEST:
    ; Code for NUM1 > NUM2 and NUM3
    MOV AH, 09H
    LEA DX, NUM1_MSG
    INT 21H

END_PROGRAM:
    MOV AX, 4C00H
    INT 21H
MAIN ENDP
END MAIN
```