The 8088 Processor Architecture

Alex Aravind

January 1, 2009

1 Introduction

Before get into specific topics, let us prepare ourselves to start the labs to go parallely. The purpose of this chapter is to introduce 8088 architecture for which you will be developing assembly level programs. Doing assembly level programming is one of the best ways to learn the computer organization and architecture.

2 Main Themes

To understand 8088 processor architecture and working principle, we need to understand:

- what are the CPU registers supported and how they are used (Register Set),
- what type of memory support is provided (Memory Support)
- what is the instruction set supported (Instruction Set), and
- what types of addressing modes are supported (Addressing Modes).

In this lecture, we discuss these questions one by one.

2.1 Register Set

The 8088 processor registers are 16 bit length, generally they are treated as 16 bits, in some cases a pair of 8 bits. Integer storage convention - little endian (low order stores lower order value).

The processor has:

- registers for the execution flow control:
  - program counter PC (also called instruction pointer (IP)) to hold the address of next instruction to be executed, and
  - condition code register CC (also called status flags (SF)) to indicate the conditions as a result of instruction executions. The registers will show the execution result status:
    - zero (Z)
    - negative (S)
    - overflow (V)
    - carry is generated (C)
    - etc.
- registers for memory access, not explicitly used at assembly program level and therefore not shown in the diagram.
- registers to maintain stack

  Stack is a data structure where operations (placing items (put) and removing items (get) are done at the top). It is mainly used to facilitate function calls. During function calls, the address of the next instruction (for the execution flow to continue after the function call returns), function parameters, local variable, etc. are maintained in the stack. The collection of data in a stack corresponding to one function is kept as contiguous and it is called a *stack frame*.

  - pointer to the top (stack pointer (SP))
  - pointer to the starting of current active part in the stack. For example, when a procedure is called, a new set of data required for that procedure (stack frame) is pushed into the stack. At a time, a stack can have many frames but usually the top frame is active. BP is used to point the starting address of current frame.
  - two other registers called source index SI and destination index DI, used to address data in the stack or to compute address in the memory.

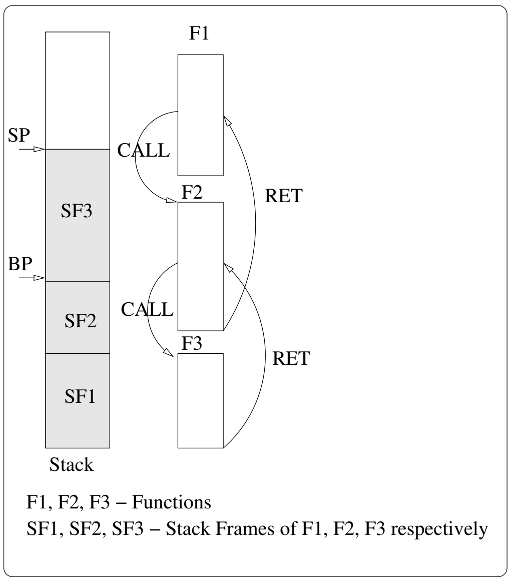

Figure 1: Stack Usage

- pointers to hold the segment addresses (memory is partitioned into segments). There are four logical segments used during the execution of a program: program code, program data, stack, other segment. Four registers are used to point the current segments for:
  - code, code segment (CS)
  - data, data segment (DS)
  - stack, stack segment (SS)
  - extra, extra segment (ES)
- general purpose registers: for arithmetic and logical operations, to hold the address (pointer) to the memory, for counter to use in loops, and other data.
  - accumulator registers, (AX)
  - base registers, (BX)
  - counter registers, (CX)
  - data registers, (DX)

  Since the content of BX can be interpreted as both a value and an address to memory location, it can be referenced either as BX or (BX), referring respectively to value and address. Note that some registers do not hold the address (for example AX). These X registers have two parts high (H) and low (L) and they can be accessed independently. For example as AH and AL.

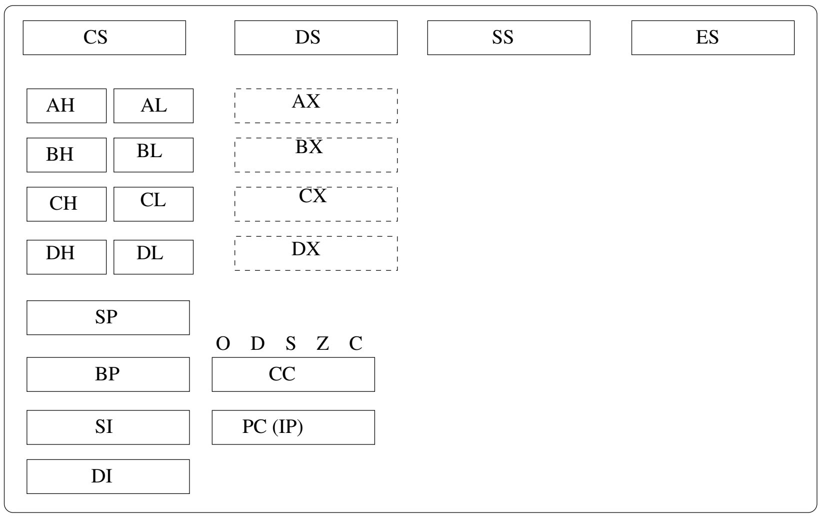

Figure 2: Registers Displayed by the Tracer

2.2 Memory Support

8088 supports 1MB, requires 20 bit word for address. But it has only 16 bit registers. To solve this problem, the memory is divided into 64KB of segments. This segment requires 16 bit register. Segment registers are used to store the segment addresses. There are 16 segments.

For example, address 672 in segment 4 has effective memory address 4 X 65536 + 672 = 262,816.

Memory addresses refers byte, in case of 2 byte word, or 4 byte long, and binary coded decimal, the consecutive addresses are accessed. The storage convention (data types) in data segment are:

- .BYTE - store each data as a byte

  Example: .BYTE 10, 20, 25. The numbers 10, 20, 25 are stored as bytes in consecutive addresses x, x+1, x+2 in the data segment. The value of x is determined based on how many other data elements are declared before.

- .WORD - store each data as a 16bit word

  Example: .WORD 10, 20, 25. The numbers 10, 20, 25 are stored as 16bit words in consecutive addresses 10 in x and x+1, 20 in x+2 and x+3, and 25 in x+4 and x+5, in the data segment.

- .LONG - store each data as a 32bit word

  Example: .LONG 10, 20, 25. The numbers 10, 20, 25 are stored as 32bit long in consecutive addresses 10 in x to x+3, 20 in x+4 to x+7, and 25 in x+8 and x+11, in the data segment.

- string

  – .ASCII - store each character in ASCII code
  – .ASCIZ - store each character in ASCII code and with a trailing zero byte.

2.3 Instruction Set

You need instructions for processing, data transfer, and execution flow control. These are the main activities involved in any data processing by a computer.

- processing

  – Arithmetic - addition, subtraction, multiplication, division, increment, decrement, shift (left and right), rotate (left and right), etc. ADD,SUB,MUL, etc.
  – Logical - AND, OR, NOT, Compare, test, etc.

- data transfer

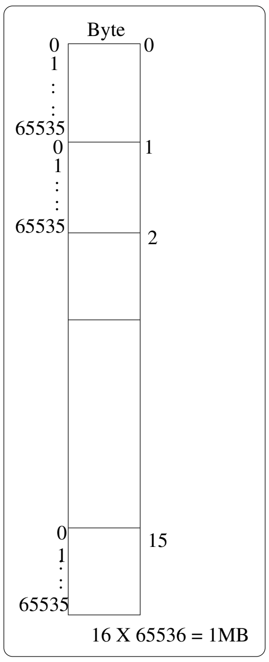

Figure 3: Memory Organization

- within CPU registers (MOV, EXHG,LEA,etc.)
- within memory,
- between memory and CPU registers (MOV, PUSH, POP, XCHG,LODS,STOS, etc.)
- on stack (PUSH, POP, and variations)

- flow control
  - to another location - jump (unconditional (JMP), conditional (for commands, refer page 721 in your text book))
  - to another routine (control needs to be returned) - (CALL and return, requires pushing values (current address, parameters, etc.) to the stack.
  - to the (operating) system (SYS)

  There are two types of jump with respect to memory management - near jump (within the segment) and far jump (outside the segment). This assembler supports only near jumps.

2.4 Addressing

Above instructions involve zero or more addresses. Addresses follow the operation code. These addresses can be classified as source address(es) and destination address(es). A location used to store constant cannot be used as a destination address. The source-destination convention in 8088 is:
opcode destination source;

Almost every instruction requires data either from memory or from CPU registers. So, some addresses refer directly to memory location(s) and some refer to register(s). The address field may contain a constant, memory address(es), and/or register name(s). Since some addressing uses addresses from some *default* registers, those names may not be explicitly specified in the address field. Push and pop have one implicit address on the stack (SP). In this scheme, since the address location is implied, they are called **implied addressing**. We briefly look at the addressing conventions used in 8088.

- **Constant:** a number - a constant value.

  Example. ADD CX, 20 - meaning the value 20 is added to CX.

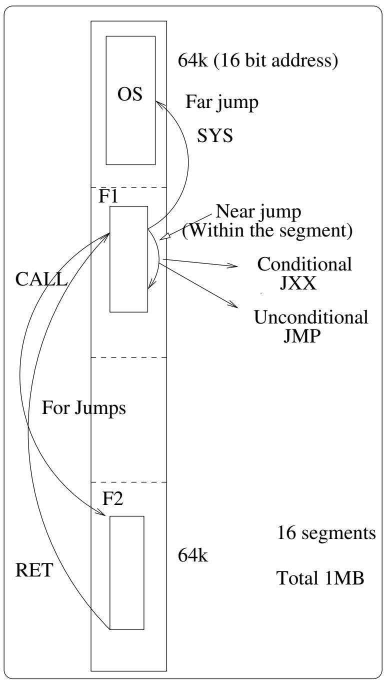

Figure 4: Flow Control

- **Memory addressing:** a constant value enclosed by parenthesis.

  Example. ADD CX, (20) - meaning add the content at location 20 and 21 to CX.

  The parenthesis indicates the content at location specified within the parenthesis. This way of directly addressing the memory is called direct addressing. In assembly programming, memory addresses are used as labels and assembler converts them into constant memory addresses during assembling.

- **Register addressing:** Only some registers can hold addresses (For example AX holds only data values.) In this scheme, since the addresses are specified implicitly through register contents they are called as (**(register) indirect addressing**). Indirect addressing is the dominant addressing scheme and based on the way the address computation is done, it has many variations:

  - **register**: - registers are specified as address containers. Addresses = register contents.

    Examples:

    mov bp, sp (Addresses = bp, sp)

    mov CX, x-y (Addresses = CX, x-p, x and y are data labels and therefore memory addresses)

    push ax (Address = ax)

    add sp, 6 (Address = sp)

  - **register displacement: -** a register and a constant are specified. Address = register content + constant.

    Examples:

    movb ah, 4(bp) (Addresses = ah, bp+4)

    mul 2(BX) (Address bx+2)

  - **register indexing -** two registers inside parentheses are specified. Address = sum of register contents.

    Example:

    push (BX)(DI) (Address = BX+DI)

  - **register with index and displacement -** two registers within a parentheses and a constant are given. Address = sum of register contents and constant.

    Example:

    NOT 25(BX)(DI) (Address = BX+DI+25)

The final source and destination addresses computed are called **effective addresses**.

Many variations are possible in register addressing. Some are:

- One address must be a register.

Addressing convention may have some implied locations (default registers) and/or some implied address computations. That is the tricky part one has to understand in order get proficiency in assembly programming. This completes a simple overview of 8088 processor, details will be learned through lab exercises.

3 The Assembler

Finally, about assembly directives. When writing an assembly language program, we need to specify different logical segments of the program for the assembler to identify. It uses three sections: TEXT section, DATA section, and BSS section (Block started by symbols) - reservation of memory in the data section for variables. This section is not initialized. Each of these sections has its own location counters. The sections indicate whether instructions to be generated or data to be generated. At the end, the linker will link the codes together in the code segment and data together in the data segment. At run time, TEXT section is stored in code segment and data and BSS segments are stored in data segment.

3.1 Labels

In assembly, labels are used for location references. Usually, programs start with some start label to indicate the beginning of the program, even if it is not used for jump purpose. For jumping (or looping), it is advised to use non-numeric labels. Numeric labels require direction (b for backward or f for forward) to be specified when directing the jump. That is if the label is 8, then LOOP 8b or LOOP 8f, depending on the direction must be specified.