Ch 5 : Assembly Language (Arithmetic and Logic Instructions)

5.1 Addition, Subtraction and Comparison

- *Addition*
  - ADD
  - memory-to-memory and segment register addition is not allowed
  - Register Addition:
    - rightmost 8 bits of flag registers and overflow flag change in these instructions to show result
  - ADD instruction modifies the contents of the sign, zero, carry, auxiliary carry, parity, and overflow flags
  - Immediate Addition:
    - for constants
  - Memory-to-Register addition:
  - Array Addition:
    - Memory arrays are sequential lists of data
  - Increment Addition:
    - Adds 1
    - Not for SRs
    - With indirect memory increments, the size of the data must be described by using the BYTE PTR, WORD PTR, DWORD PTR, or QWORD PTR directives.
      - This is because assembler program cannot determine the size for which the INC instruction is given
    - Affects the flag bits but not the carry flag
      - Because programs often depend on carry flag's contents
    - Prefer to use ADD to increment by more than 1
  - Addition-with-Carry:
    - ADC adds the bit in the carry flag (C)to the operand data.
    - for softwares dealing with numbers wider than 16 bits
    - the most significant 16 bits of this addition are added with the carry flag
  - Exchange and Add for the 80486–Core2 Processors:
    - XADD
    - for 80486 and above
    - adds the source to the destination and stores the sum in the destination
    - after the addition takes place, the original value of the destination is copied into the source operand.
    - works with any register size and any memory operand
- *Subtraction*
  - SUB
  - Register Subtraction:
  - Immediate Subtraction:
  - Both carry flags , C and A hold the borrow
  - An overflow can occur if signed result is not in the range [+127, -128]
  - Decrement Subtraction:
    - Also requires directives for memory size
  - Subtraction-with-Borrow (SBB) :
    - the borrow stored in C is also subtracted from the difference
    - for wider than 16 bit subtractions
    - Wide subtractions require that borrows propagate through the subtraction, just as wide additions propagate the carry.
- *Comparison*
  - CMP
  - Only the flag bits change not destination operand
  - useful for checking the entire contents of a register or a memory location against another value
  - followed by a conditional jump
    - checks the condition of the flag bits
    - JA: jump above (if greater)
    - JB: jump below
    - JAE: if equal or above
    - JBE: if equal or below
    - e.g. CMP AL, 10H
      - If AL is greater, JA

- Compare and Exchange (80486–Core2 Processors Only):
  - CMPXCHG
  - compares the destination operand with the accumulator.
  - If they are equal, the source operand is copied into the destination;
  - if they are not equal, the destination operand is copied into the accumulator.
  - CMPXCHG8B for Pentium Core 2
    - for quad words
    - CMPXCHG16B for 64 bit mode
  - The Z (zero) flag bit indicates that the values are equal after the comparison.
  - This instruction has a bug that will cause the operating system to crash.

5.2 Multiplication and Division

- *Multiplication*
  - MUL(unsigned) / IMUL(signed)
  - product is double-width
    - 8 bit multiplied with 8 bit is 16 bit
    - same for 16, 32 and 64
  - Overflow and carry flags change
    - if the most significant 8 bits of the result are zero, both C and O flag bits equal zero
    - These flag bits show that the result is 8 bits wide(C=0) or 16 bits wide (C=1).
    - Same for 16, 32, 64
  - 8-bit Multiplication:
    - multiplicand in AL
    - multiplier in register or memory location
    - product is stored in AX
    - Immediate only allowed for special signed
    - For signed multiplication,
      - the product is in binary form, if positive
      - in two's complement form, if negative.
  - 16-bit multiplication:
    - AX contains the multiplicand
    - DX-AX contains the product
      - DX: most significant 16 bits
      - AX: least significant 16 bits
  - A Special Immediate 16-Bit Multiplication:
    - 80186 and above
    - must be signed
    - contains 3 operands
      - destination register
      - source register/memory location
      - immediate data
    - E.g. MUL CX,DX,12H
  - 32-bit multiplication:
    - The product (64 bits wide) is found in EDX–EAX, where EAX contains the least significant 32 bits of the product.
  - 64-bit multiplication:
    - product appears in RDX:RAX register pair
- *Division*
  - IDIV(signed) / DIV(unsigned)
  - dividend is double width
  - can result in two different types of errors;
    - an attempt to divide by zero
    - a divide overflow.
      - when a small number divides into a large number
    - the microprocessor generates an interrupt if a divide error occurs
  - 8-bit division:
    - 16 bit number is divided by 8 bit
    - AX: dividend (before division)
    - AL: quotient (after division)

        - AH: remainder (after division)
        - remainder assumes sign of the dividend and is always an integer.
        - To convert one number to 16 bit in AX,
            - The most significant 8 bits of unsigned numbers are zero extended (cleared to 0)
                - using MOVZX in 80386
            - The least 8 bits of signed numbers must be sign extended into the most significant 8 bits.
                - a special instruction sign-extends AL into AH. The CBW (convert byte to word) instruction performs this conversion.
                - MOVSX in 80386
    - 16-bit division:
        - DX-AX contains the dividend (32 bit)
        - AX: quotient
        - DX: remainder
        - one number is converted to 32 bits
            - If a 16-bit unsigned number is placed in AX, DX must be cleared to zero.
                - In the 80386 and above, the MOVZX instruction is used.
            - If AX is a 16-bit signed number, the CWD (convert word to doubleword) instruction sign-extends it into a signed 32-bit number.
                - In the 80386 and above is available, the MOVSX instruction is used.
    - 32-bit division:
        - EDX-EAX contains the dividend (64 bits)
        - EAX : quotient
        - EDX : remainder
        - functions the same:
            - The CDQ (convert doubleword to quadword) instruction is used before a signed division to convert the 32-bit contents of EAX into a 64-bit signed number in EDX–EAX.
    - The Remainder:
        - The remainder could be used to round the quotient or just dropped to truncate the quotient.
        - If the division is unsigned, rounding requires that the remainder be compared with half the divisor to decide whether to round up the quotient.
        - The remainder could also be converted to a fractional remainder.
    - 64-bit division:
        - RDX-RAX contains the dividend
        - RAX : quotient
        - RDX : remainder

5.3 BCD and ASCII Arithmetic

- BCD: Binary-coded decimal
    - Systems like point-of-sales (cash registers)
- ASCII : American Standard Code for Information Interchange
- *BCD Arithmetic*
    - Manipulation only done on AL
    - DAA (Decimal Adjust after Addition):
        - Follows ADD or ADC to adjust the result into a BCD result.
        - the addition must be 8 bits at a time (because AL)
        - 1234BCD = 1234H
    - DAS (Decimal Adjust after Subtraction):
        - Follows SUB or SBB
- *ASCII Arithmetic*
    - 30H to 39H for 0-9
    - Use AX as source and destination
    - AAA (ASCII Adjust after Addition):
        - Clears AH if result is less than 10 and adds 1 to AH if greater to get the right result
        - You add 3030H to result to get final answer.
    - AAD (ASCII Adjust after Division):
        - requires that the AX register contain a two-digit unpacked BCD number (not ASCII) before executing.
            - e.g. 72H is written as 0702H which will be converted to 0048H by AAD
        - After adjusting the AX register with AAD, it is divided by an unpacked BCD number to generate a single-digit result in AL with any remainder in AH.

- AAM (ASCII Adjust after Multiplication):
  - E.g. multiplying 5 by 5, we'll get 0019H, adjusted to get 0205H, added 3030H to get 3235H to get final output
  - Converts by dividing AX by 10
    - This can be changed by storing 0BH instead of 0AH in the second instruction.
    - Use DB 0D4H, 0BH instead of AAM for 11 (0BH)
  - Add 3030H for final output
- AAS (ASCII Adjust after Subtraction):
  - When 35H subtracts from 39H,
    - The result will be 04H, which requires no correction. Here, AAS will modify neither AH nor AL.
  - If 38H is subtracted from 37H,
    - AL will equal 09H and the number in AH will decrement by 1.
    - This decrement allows multiple-digit ASCII numbers to be subtracted from each other.

5.4 Basic Logic Instructions

- provide binary bit control in low-level software.
- Allow bits to be set, cleared, or complemented.
- Low-level software appears in machine language or assembly language form and often controls the I/O devices in a system.
- All logic instructions affect the flag bits.
- Logic operations always clear the carry and overflow flags, while the other flags change to reflect the condition of the result.
- Right most bit is bit 0
- Saves money to replace the gates by these instructions in embedded applications
- *AND*
  - Logical multiplication
  - Can replace AND gate if not for speed
  - Clears bits of a binary number
  - The task of clearing a bit in a binary number is called masking.
  - Uses any addressing mode except memory-to-memory and segment-register
  - ASCII-coded numbers can be converted to BCD by ANDing to mask off the leftmost 4 bits
- *OR*
  - Logical addition
  - Inclusive-OR
  - Uses any addressing mode except segment-register
  - Sets any bit
  - Converts BCD to ASCII
- *Exclusive-OR*
  - XOR
  - 1 only if both inputs are different
  - Inverts or complements the bits
    - XOR BX, 0FFC0H inverts only the leftmost 10 bits not rightmost 6 bits
  - Commonly used to clear a register to 0
    - XOR CH, CH clears CH
      - requires only 2 bytes of memory
    - MOV CH, 00H also clears CH
      - requires 3 bytes of memory
- *Test and Bit test instructions*
  - TEST:
    - Performs AND
    - AND changes the destination operand but not TEST
    - Only affects the condition of flag register that indicates the result of the test
    - Works same way as CMP
      - but tests only a single bit unlike CMP
    - Z=1 if the bit under test is 0, Z=0 if bit under test is not zero
    - Followed by JZ or JNZ
    - destination operand is tested against immediate data
    - Value of immediate data is 1 to test the rightmost bit position, 2 to test the next bit, 4 for the next, and so on.
  - Bit Test:
    - Test the bit position in the destination operand selected by the source operand
      - E.g. BT AX, 4 tests bit position 4 in AX

        - Result is located in carry flag bit
            - carry set if the position is 1 else cleared.
        - *Table 5-20 on page 201 of pdf* for other kinds of bit test
        - E.g. The BTC AX,4 instruction complements bit position 4 after testing it,
            - the BTR AX,4 instruction clears it (0) after the test,
            - the BTS AX,4 instruction sets it (1) after the test.
- *NOT and NEG*
    - NOT:
        - Logical inversion or one's complement
        - inverts all bits of a byte, word or doubleword
        - considered logical operation
    - NEG:
        - Arithmetic sign inversion or two's complement
        - changes the arithmetic sign of the signed number
        - Considered arithmetic operation

5.5 Shift and Rotate

- Manipulate binary numbers at the binary bit level
- low-level software use to control I/O devices
- *Shift:*
    - *SHL, SHR, SAL, SAR*
    - positions or moves numbers to the left or right within a register or memory location.
    - performs simple arithmetic such as multiplication by powers of (left shift) and division by powers of (right shift).
    - Logical shifts move 0 into rightmost bit positions for left and leftmost bits for right
        - work with unsigned numbers
    - Arithmetic
        - left shift is similar to logical left
        - right shift is different because the arithmetic right shift copies the sign-bit through the number, whereas the logical right shift copies a 0 through the number.
        - Work with signed numbers
    - left multiplies by 2 and right divides by 2 for each position shifted
    - can either use immediate shift count or use CL to hold shift count
        - When CL is the shift count, it doesn't change on execution of shift.
        - It is a modulo 32 count i.e. shift 33 will only shift 1
    - shifting and adding instead of multiplying can sometimes work faster
    - Double-Precision Shifts (80386–Core2 Only):
        - SHLD and SHRD
        - three operands
        - SHRD AX, BX, 12 logically shifts AX right by 12 positions and the rightmost 12 bits of BX are shifted into leftmost 12 bits of AX. BX doesn't change.
- *Rotate:*
    - *RCL, ROL, RCR, ROR*
    - positions binary data by rotating the information in a register or memory location, either from one end to another or through the carry flag.
    - They are often used to shift or position numbers that are wider than 16 bits in the 8086–80286 microprocessors or wider than 32 bits in the 80386 through the Core2.
    - Rotate through carry flag or register/ memory location
    - Same addressing modes as shift
    - rotate count can be immediate or located in register CL.
    - RCL rotates carry into BX and its leftmost into carry
- *Bit Scan instructions:*
    - Scan through a number searching for a 1-bit.
    - This is accomplished by shifting the number
    - BSF (bit scan forward) and BSR (bit scan reverse)
    - BSF scans from leftmost, BSR scans from rightmost
    - Z = 0 if 1 is found and its position is placed into destination operand

5.6 String Comparisons

- string instructions are very powerful because they allow the programmer to manipulate large blocks of data with relative ease.

- Block data manipulation occurs with the string instructions MOVS, LODS, STOS, INS, and OUTS.
- *SCAS (String Scan instruction):*
  - Compares AL, AX or EAX
  - Subtracts memory without affecting either.
  - SCASB for byte, SCASW for word, SCASD for doubleword
  - the contents of the extra segment memory location addressed by DI are the ones compared
  - Use the D flag to select either auto-increment or -decrement for DI.
  - Also repeat if prefixed to
    - REPNE SCASB (for repeat while not equal)
    - REPE SCASB (for repeat while equal)
- *CMPS (Compare string instruction)*
  - Compares two sections of memory data
  - Bytes - CMPSB, words - CMPSW, doublewords - CMPSD, quadwords - CMPSQ
  - The contents of the data segment memory location addressed by SI are compared with the contents of the extra segment memory location addressed by DI.
  - increments of decrements both DI and SI
  - Normally used with the repeat prefix.