5 Assembly Language Programming (2)

Arithmetic Instruction

- **Addition**: *CPU will treat all the values as unsigned value*.

  ```
  ADD dest, src ;dest = dest + src
  ```

  - `dest` can be a register or in memory;
  - `src` can be a register, in memory or immediate;
  - No mem-to-mem operations in 8086;
  - Change `ZF`, `SF`, `AF`, `CF`, `OF`, `PF`.

  ```
  ADC dest, src ;dest = dest + src + CF
  ```

  - For multi-byte numbers;
  - If there is a carry from last addition, adds 1 to the result;
  - Change `ZF`, `SF`, `AF`, `CF`, `OF`, `PF`.

  ```
  INC dest ;dest = dest + 1
  ```

  - `dest` can be a register or in memory;
  - `dest` cannot be an immediate;
  - Change `ZF`, `SF`, `AF`, `OF`, `PF`; *DOES NOT* change `CF`.
- **Subtraction**: *CPU will treat all the values as unsigned value*.

  ```
  SUB dest, src ;dest = dest - src
  ```

  - `dest` can be a register or in memory;
  - `src` can be a register, in memory or an immediate
  - No mem-to-mem operations in 8086;
  - Change `ZF`, `SF`, `AF`, `CF`, `OF`, `PF`.

  ```
  SBB dest, src ;dest = dest - src - CF
  ```

  - For multi-byte numbers;
  - If there is a borrow from last subtraction, subtracts 1 from the result.
  - Change `ZF`, `SF`, `AF`, `CF`, `OF`, `PF`.

  ```
  DEC dest ;dest = dest - 1
  ```

  - Destination can be a register or in memory;
  - Destination cannot be an immediate;
  - Change `ZF`, `SF`, `AF`, `OF`, `PF`; *DOES NOT* change `CF`.

  How to implement subtraction

  - take the 2's complement of the `src`;
  - add it to the `dest`;

  - *invert* the carry.
- **Multiplication**:

  Unsigned multiplication

  ```
  MUL operand
  ```

  - Change `OF`, `CF`; Unpredictable: `SF`, `ZF`, `AF`, `PF`.
  - byte * byte: one implicit operand is `AL`, the other is `operand`, result is stored in `AX`;
  - word * word: one implicit operand is `AX`, the other is `operand`, result is stored in `DX` and `AX`;
  - word * byte: `AL` hold the byte and `AH = 0`, the word is the operand, result is stored in `DX` and `AX`.
- **Division**:

  Unsigned Division

  ```
  DIV denominator
  ```

  - Unpredictable: `OF`, `CF`, `SF`, `ZF`, `AF`, `PF`.
  - denominator cannot be zero; quotient（商） cannot be too large for the assigned register;
  - denominator can be in a register or in memory;
  - byte / byte: numerator in `AL`, clear `AH`; quotient in `AL`, remainder in `AH`;
  - word / word: numerator in `AX`, clear `DX`; quotient in `AX`, remainder in `DX`;
  - word / byte: numerator in `AX`; quotient in `AL` (max `0FFFH`), remainder in `AH`;
  - double-word / word: numerator in `DX` and `AX`, quotient in `AX` (MAX `0FFFFH`), remainder in `DX`.

Logical Instructions

- **Bitwise operations**

  ```
  AND dest, src
  OR dest, src
  XOR dest, src
  ```

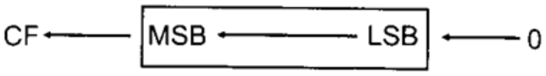
  - `dest` can be a register or in memory; `src` can be a register, in memory, or immediate;
  - Update `SF`, `ZF`, `PF`; `AF` is undetermined;
  - Clear `CF` and `OF` (set to zero).

  ```
  NOT operand
  ```

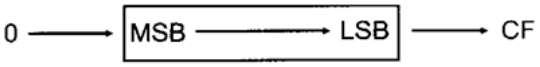
  - `operand` can be a register or in memory;
  - DOES NOT change the flag register.
- **Logical SHIFT**

  ```
  SHL dest, times
  SHR dest, times
  ```

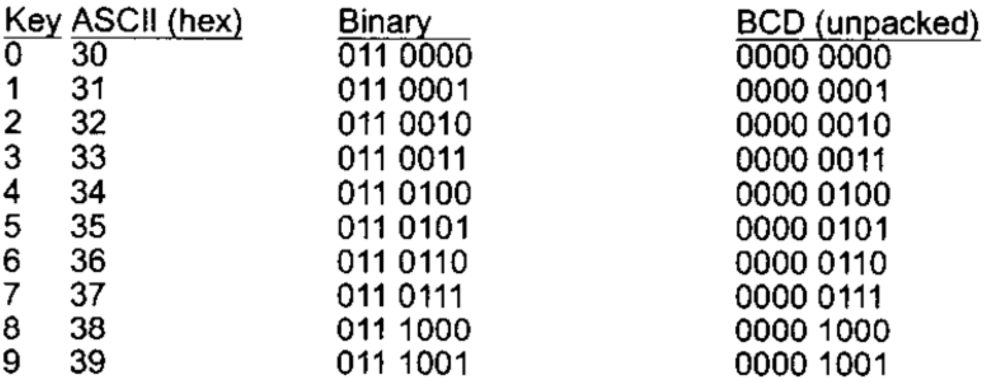
  - Left shift:



  - Right shift:



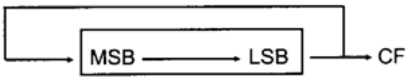
  - `dest` can be a register or in memory;
  - `CF` will be updated by the last out-of-range bit.
  - if `times = 1`, we can write `SHR xx, 1`; else we have to write `MOV CL, times` and `SHR xx, CL`. (`SHL` similarly)
  - Put zero in the shifted bits.

> [*Example*] Application of shifts
>
> 
>
> | Key | ASCII (hex) | Binary | BCD (unpacked) |
> |---|---|---|---|
> | 0 | 30 | 011 0000 | 0000 0000 |
> | 1 | 31 | 011 0001 | 0000 0001 |
> | 2 | 32 | 011 0010 | 0000 0010 |
> | 3 | 33 | 011 0011 | 0000 0011 |
> | 4 | 34 | 011 0100 | 0000 0100 |
> | 5 | 35 | 011 0101 | 0000 0101 |
> | 6 | 36 | 011 0110 | 0000 0110 |
> | 7 | 37 | 011 0111 | 0000 0111 |
> | 8 | 38 | 011 1000 | 0000 1000 |
> | 9 | 39 | 011 1001 | 0000 1001 |
>
> Notice that the last 4 bits of ASCII are exactly the same as the original number.
>
> - **ASCII to unpacked BCD**
>
> ```
> asc DB '3'
> unpack DB ?
> ;----------------------
> MOV AH asc
> AND AH 0FH  ; to clear the high bits;
> MOV unpack AH ; get the value
> ```
>
> - **ASCII to packed BCD**
>
> ```
> asc DB '23'
> pack DB ?
> ;-----------------------
> MOV AH asc
> MOV AL asc+1
> AND AX, 0F0FH
> MOV CL, 4
> SHL AH, CL  ; shift to the right place
> OR AH, AL   ; combine together
> MOV pack, AL
> ```

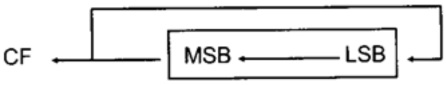
- **Rotate Shift**

```
ROL dest, times
ROR dest, times
```

- Left rotate shift:

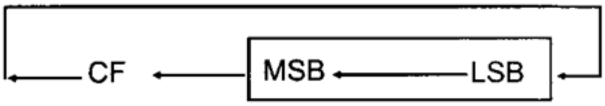
- Right rotate shift:

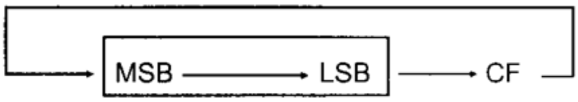
- `dest` can be a register, in memory;
- The `CF` will be updated;

```
RCL dest, times
RCR dest, times
```

- Left rotate carry shift

- Right rotate carry shift

- `dest` can be a register, in memory;
- The `CF` is shifted with the other bits;

Unsigned Compare Instruction

```
CMP dest, src
```

- Flags affected as `dest - src` but operands remain unchanged.
- When combining with jump instruction, we mainly focus on the `CF` and `ZF` value.
  - `dest > src`: `CF == 0, ZF == 0`;
  - `dest = src`: `CF == 0, ZF == 1`;
  - `dest < src`: `CF == 1, ZF == 0`.
- Related jumps:
  - `JA` (Jump above, also `JNBE`)
  - `JB` (Jump below, also `JNAE`)
  - `JAE` (Jump above or equal, also `JNB`)
  - `JBE` (Jump below or equal, also `JNA`)

Signed Compare Instruction

```
CMP dest, src
```

- *The same instruction as* unsigned compare instruction. Flags affected as `dest - src` but operands remain unchanged.
- When combining with jump instruction, we mainly focus on the `CF` and `ZF` value.
  - `dest > src`: `OF == SF and ZF == 0`;

    - `dest = src`: `ZF == 1`;
    - `dest < src`: `OF != SF`.
- Related jumps:
    - `JG` (Jump greater, also `JNGE`)
    - `JL` (Jump less, also `JNLE`)
    - `JGE` (Jump greater or equal, also `JNL`)
    - `JLE` (Jump less or equal, also `JNG`)

Unsigned vs Signed Number

- Execution: treated as unsigned numbers;
- Interpretation: `CF` is updated by treating both numbers as unsigned, `OF` is updated by treating both numbers as signed.