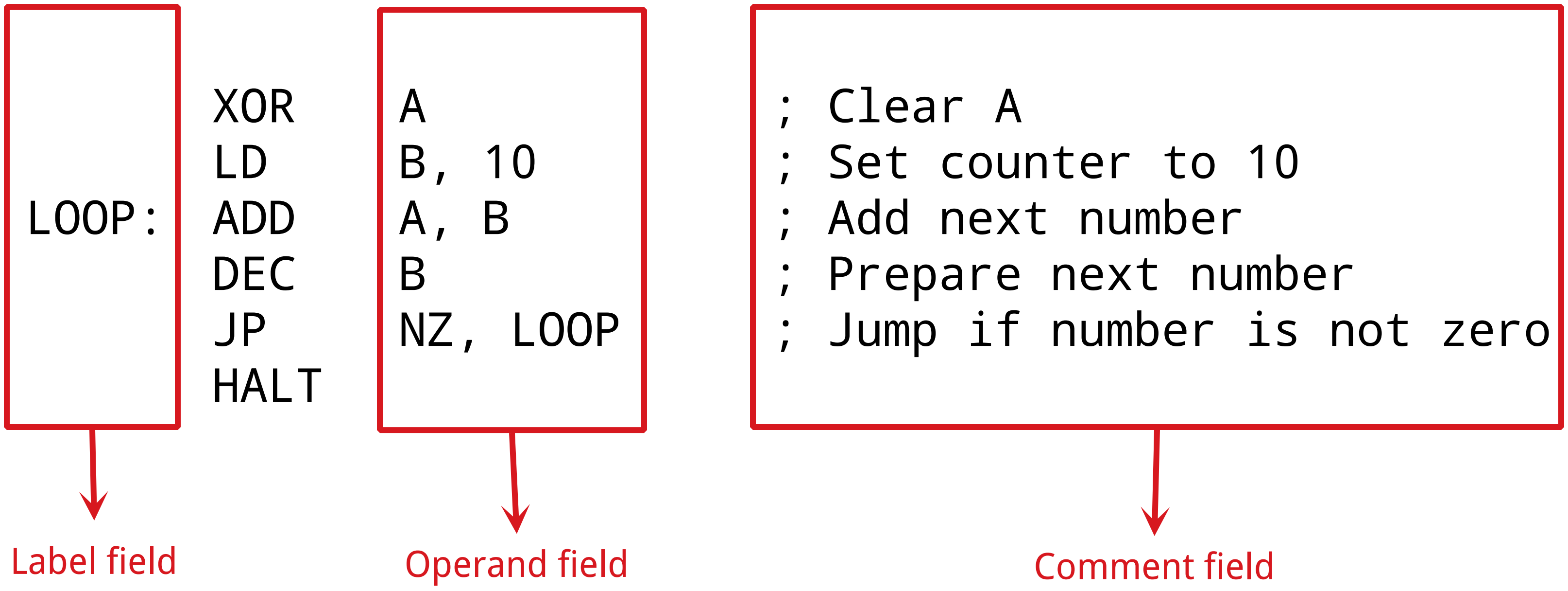
Microprocessors, Lecture 4

Z80 Assembly Language

Hamid Fadishei
Assistant Professor, University of Bojnord

Spring 2015

Assembly source structure



```
        XOR   A          ; Clear A
        LD    B, 10      ; Set counter to 10
LOOP:   ADD   A, B       ; Add next number
        DEC   B          ; Prepare next number
        JP    NZ, LOOP   ; Jump if number is not zero
        HALT
```

Label field

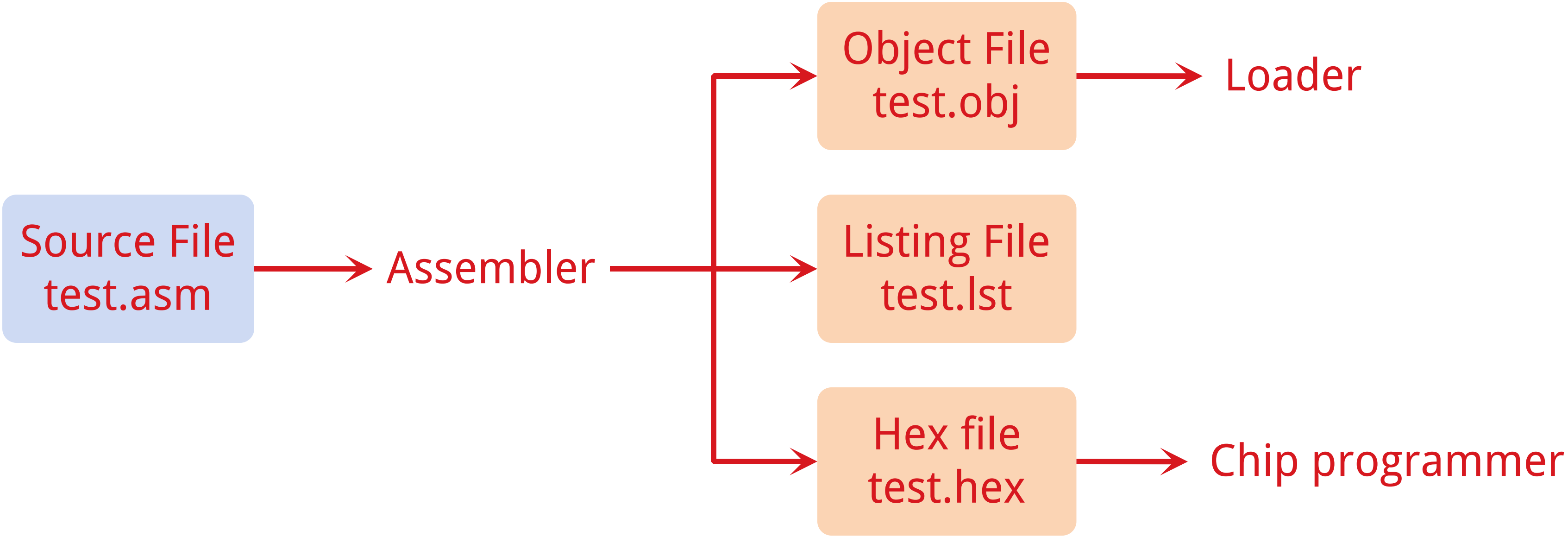
Operand field

Comment field

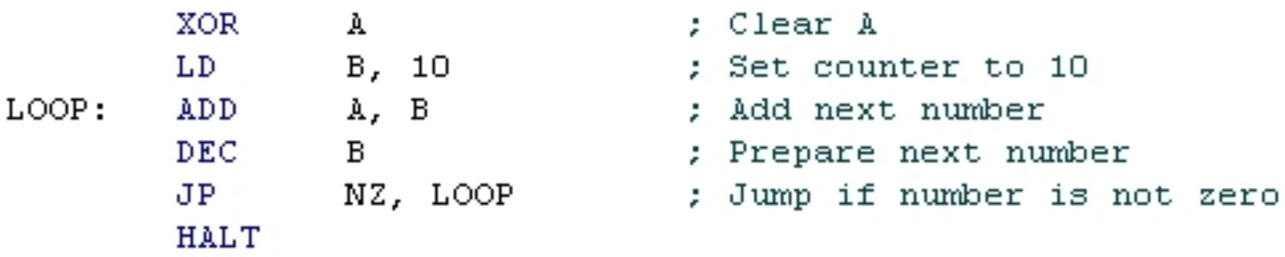
Some definitions

- Assembler
  - Converts the assembly program source code to executable machine code
    - Cross-assembler
      - Runs on a system other than the one you are programming for
    - Resident assembler (self-assembler)
      - Runs on a computer for which it assembles programs
- Two-pass assembler
  - Goes through the source twice
  - Sometimes only one pass is not enough to assemble
  - Example: Labels which are used before definition
- Loader
  - Takes the output of assembler (object code) and puts it into volatile memory (e.g. into SRAM)
- Programmer
  - Takes the output of assembler (hex code) and writes it into permanent memory (e.g. into FLASH)
- Pseudo operations
  - Do not directly translate to a machine instruction, but help the assembler to do its work
  - ORG, DB, DW, EQU, …

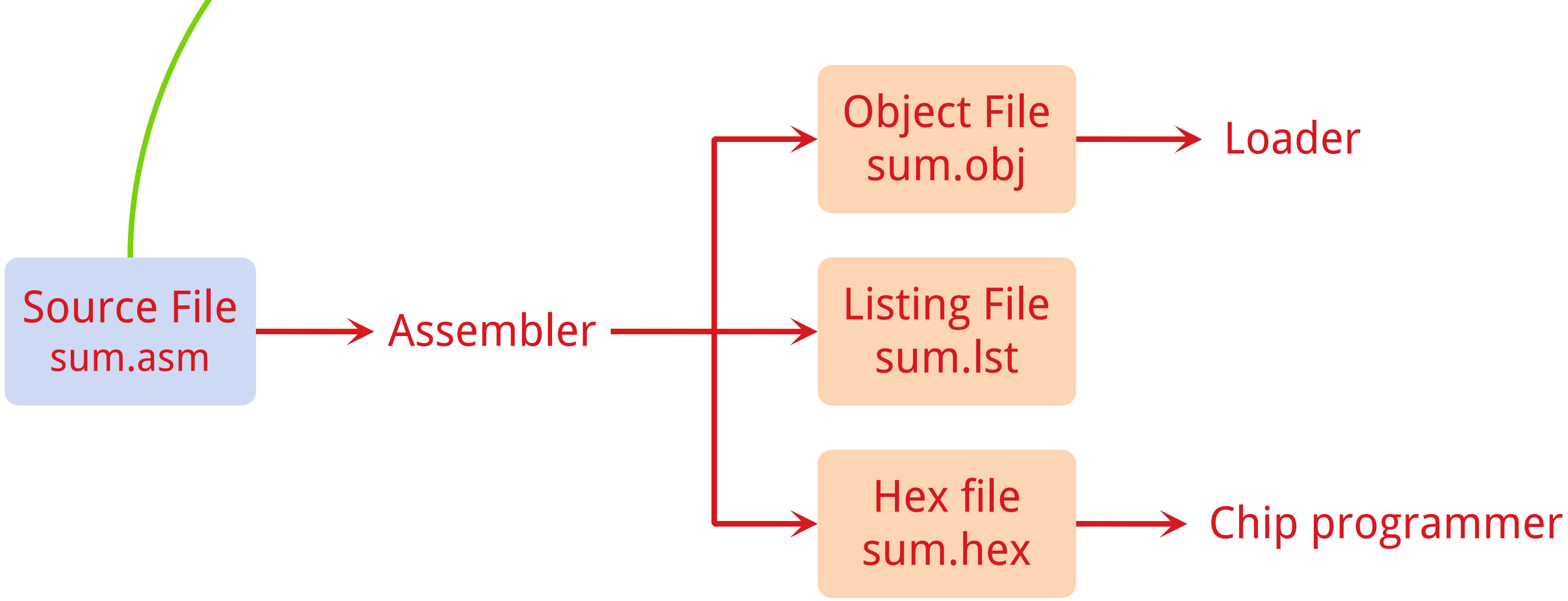
Assembly programming



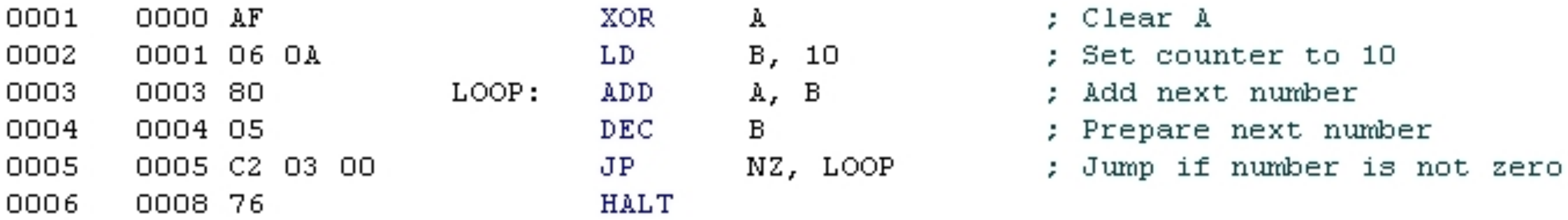
Assembly programming



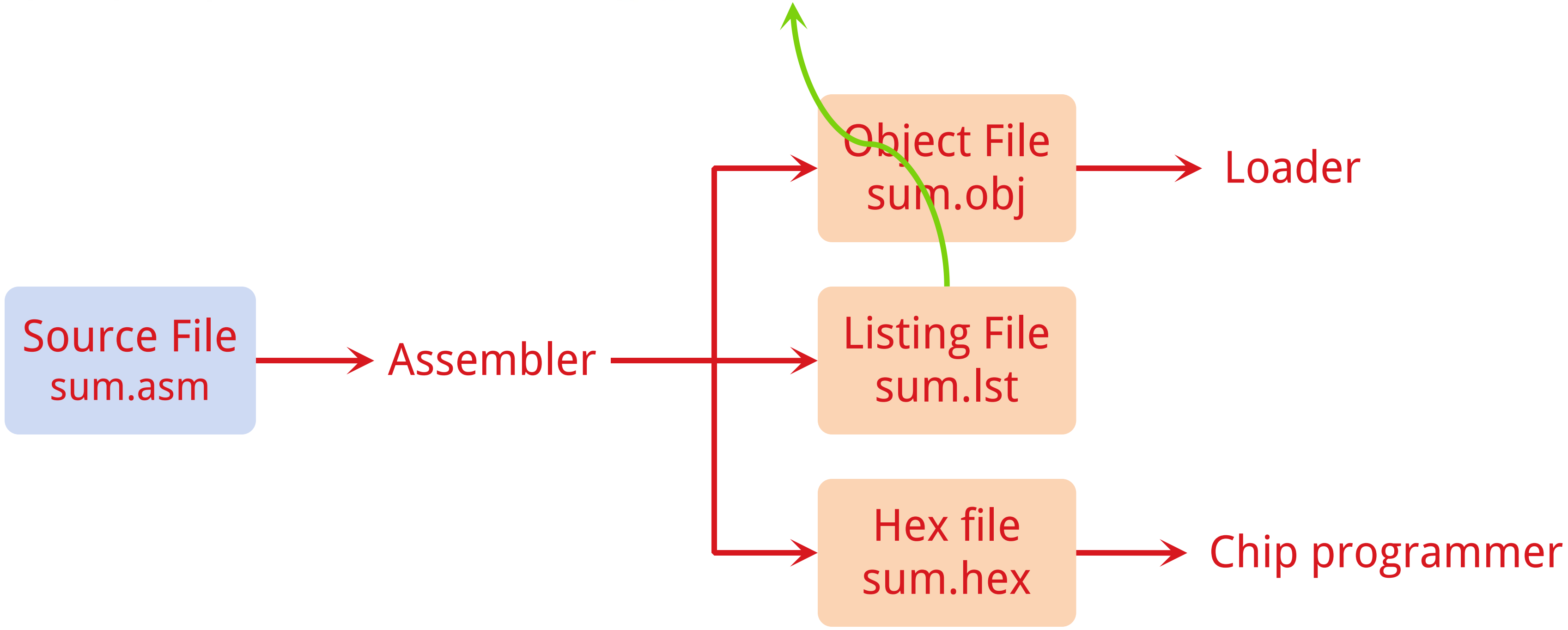
```
        XOR     A               ; Clear A
        LD      B, 10           ; Set counter to 10
LOOP:   ADD     A, B            ; Add next number
        DEC     B               ; Prepare next number
        JP      NZ, LOOP        ; Jump if number is not zero
        HALT
```



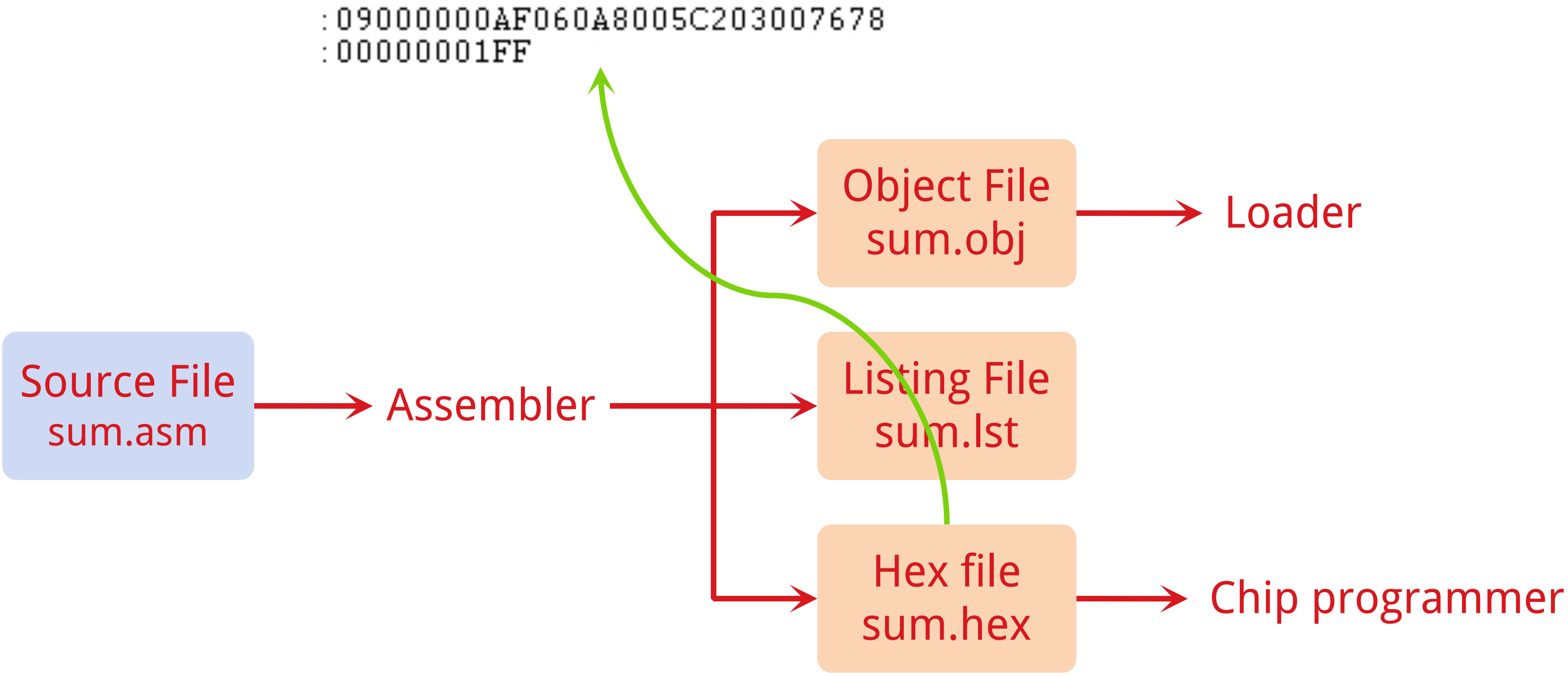
Assembly programming



```
0001    0000 AF                  XOR    A          ; Clear A
0002    0001 06 0A               LD     B, 10      ; Set counter to 10
0003    0003 80          LOOP:   ADD    A, B       ; Add next number
0004    0004 05                  DEC    B          ; Prepare next number
0005    0005 C2 03 00            JP     NZ, LOOP   ; Jump if number is not zero
0006    0008 76                  HALT
```



Assembly programming



```
:09000000AF060A8005C203007678
:00000001FF
```

Source File
sum.asm

Assembler

Object File
sum.obj

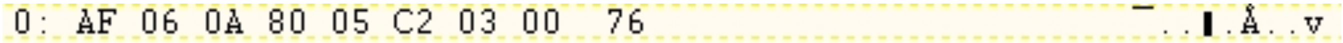
Listing File
sum.lst

Hex file
sum.hex

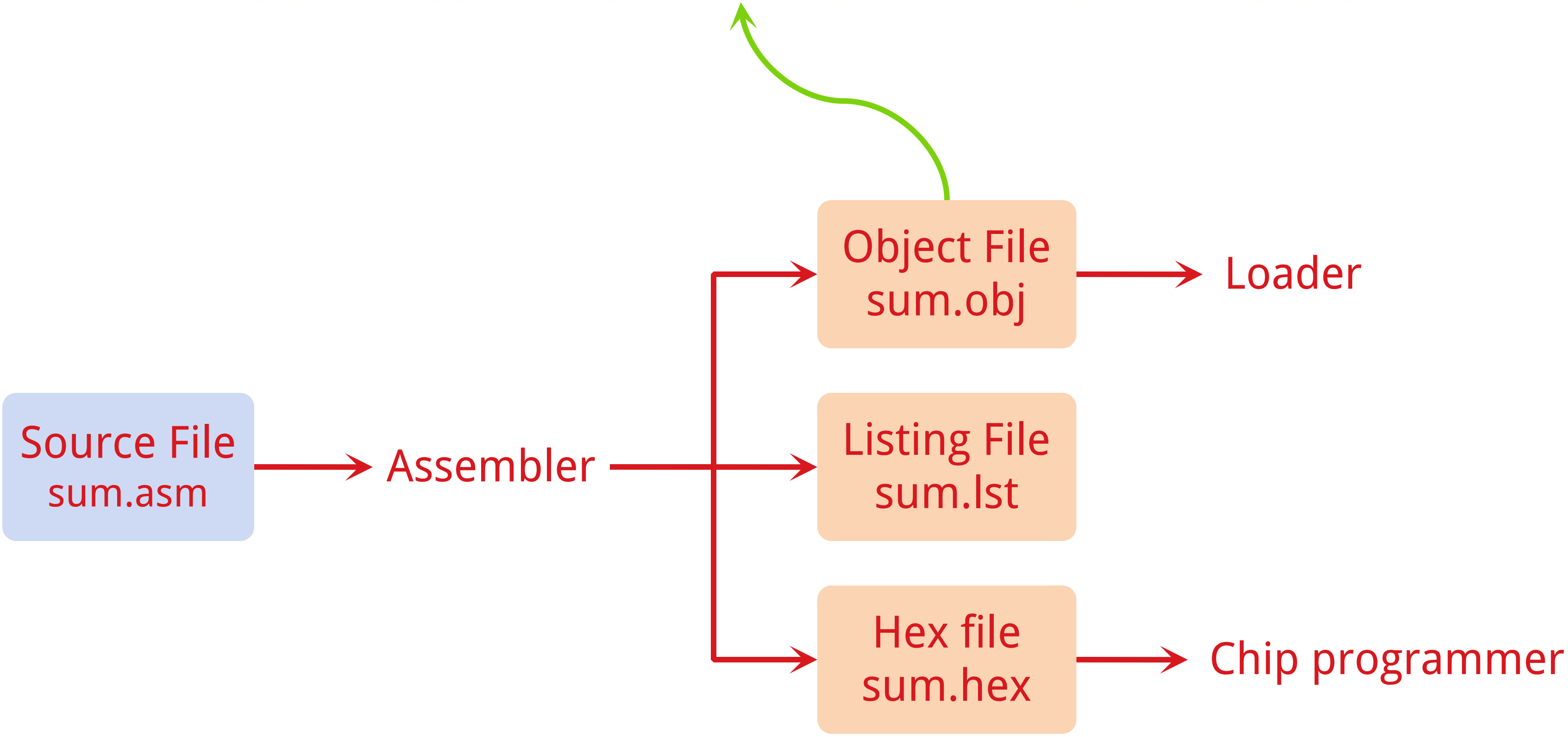
Loader

Chip programmer

Assembly programming



```
0:  AF 06 0A 80 05 C2 03 00   76                                    ¯..▌.Å..v
```

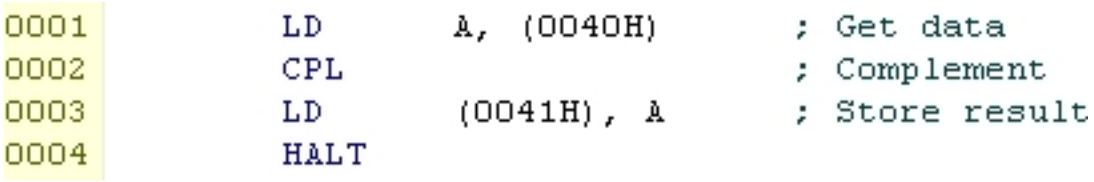


Do we have to learn assembly?

- Today microprocessors can be programmed high-level
  - In old days, hand-writing assembly code usually gave more efficient programs
  - But today high-level compilers are very intelligent
  - Learning low-level programming concepts is still necessary
    - A base without which we may not fully understand concepts
    - You may need to reverse-engineer a machine code
    - You might not find a high level compiler for your microprocessor

Programming patterns: Basics

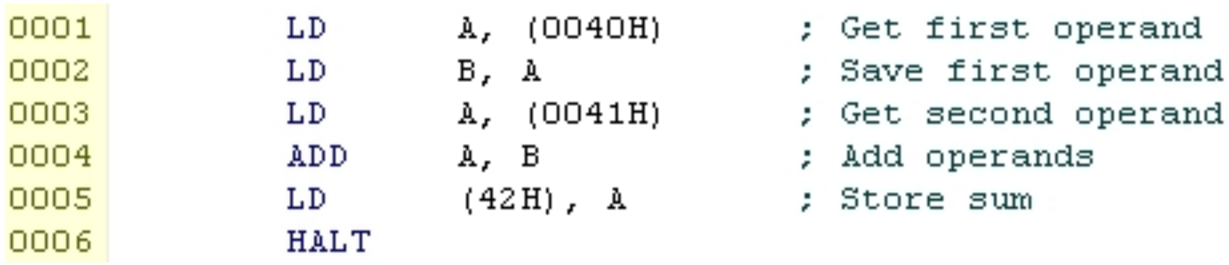
- Example: 1’s complement
  - Logically complement the contents of memory location 0040H and place the result into memory location 0041H
- Solution



```
            LD      A, (0040H)          ; Get data
            CPL                         ; Complement
            LD      (0041H), A          ; Store result
            HALT
```

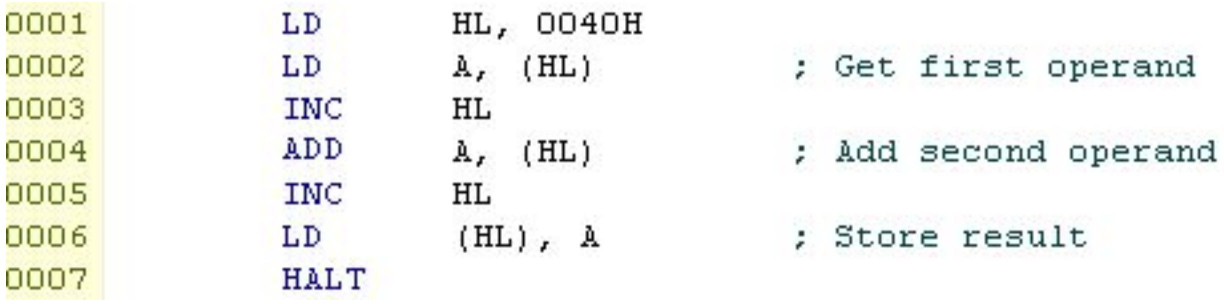
Programming patterns: Basics

- Example: 8-bit addition
  - Add the contents of memory location 0040H and 0041H, and place the result into memory location 0042H
- Solution 1



```
        LD      A, (0040H)      ; Get first operand
        LD      B, A            ; Save first operand
        LD      A, (0041H)      ; Get second operand
        ADD     A, B            ; Add operands
        LD      (42H), A        ; Store sum
        HALT
```

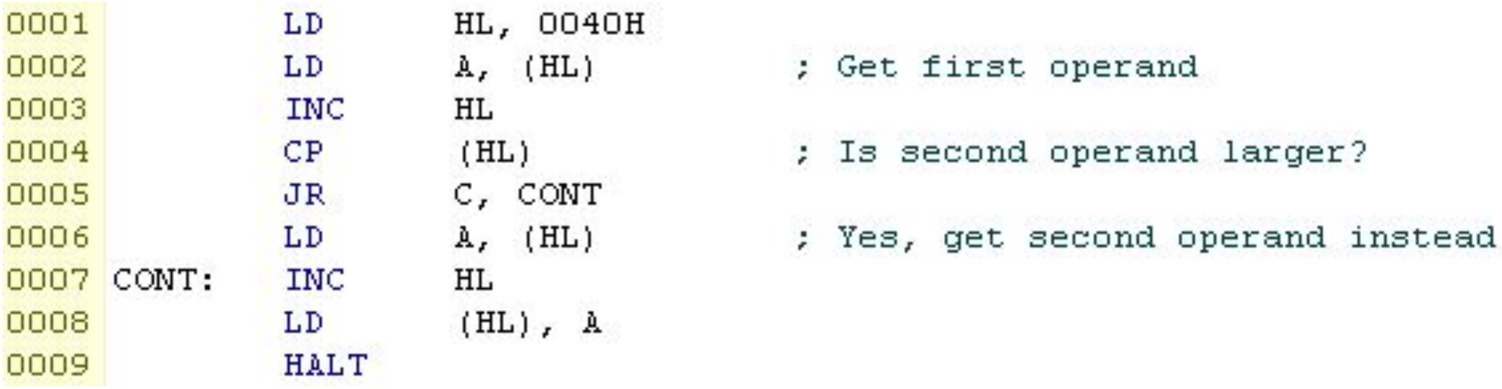
- Solution 2



```
        LD      HL, 0040H
        LD      A, (HL)         ; Get first operand
        INC     HL
        ADD     A, (HL)         ; Add second operand
        INC     HL
        LD      (HL), A         ; Store result
        HALT
```

Programming patterns: Basics

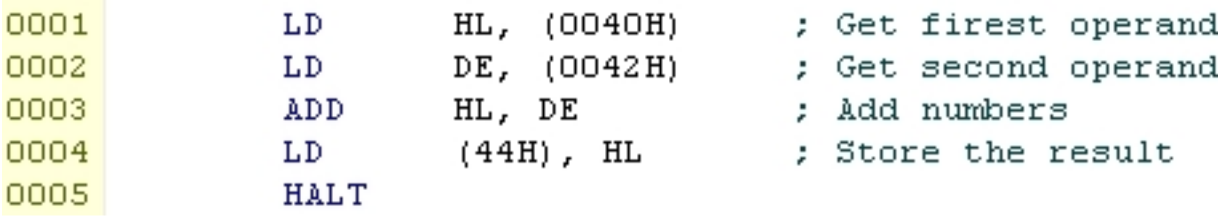
- Example: max(m,n)
  - Place the larger of the contents of memory locations 0040H and 0041H into memory location 0042H.
- Solution



```
        LD      HL, 0040H
        LD      A, (HL)         ; Get first operand
        INC     HL
        CP      (HL)            ; Is second operand larger?
        JR      C, CONT
        LD      A, (HL)         ; Yes, get second operand instead
CONT:   INC     HL
        LD      (HL), A
        HALT
```

Programming patterns: Basics

- Example: 16-bit addition
  - Add the 16-bit number in memory location 0040H and 0041H to the number in memory location 0042H and 0043H. Store the result in 0044H and 0045H.
- Solution



```
        LD      HL, (0040H)     ; Get firest operand
        LD      DE, (0042H)     ; Get second operand
        ADD     HL, DE          ; Add numbers
        LD      (44H), HL       ; Store the result
        HALT
```

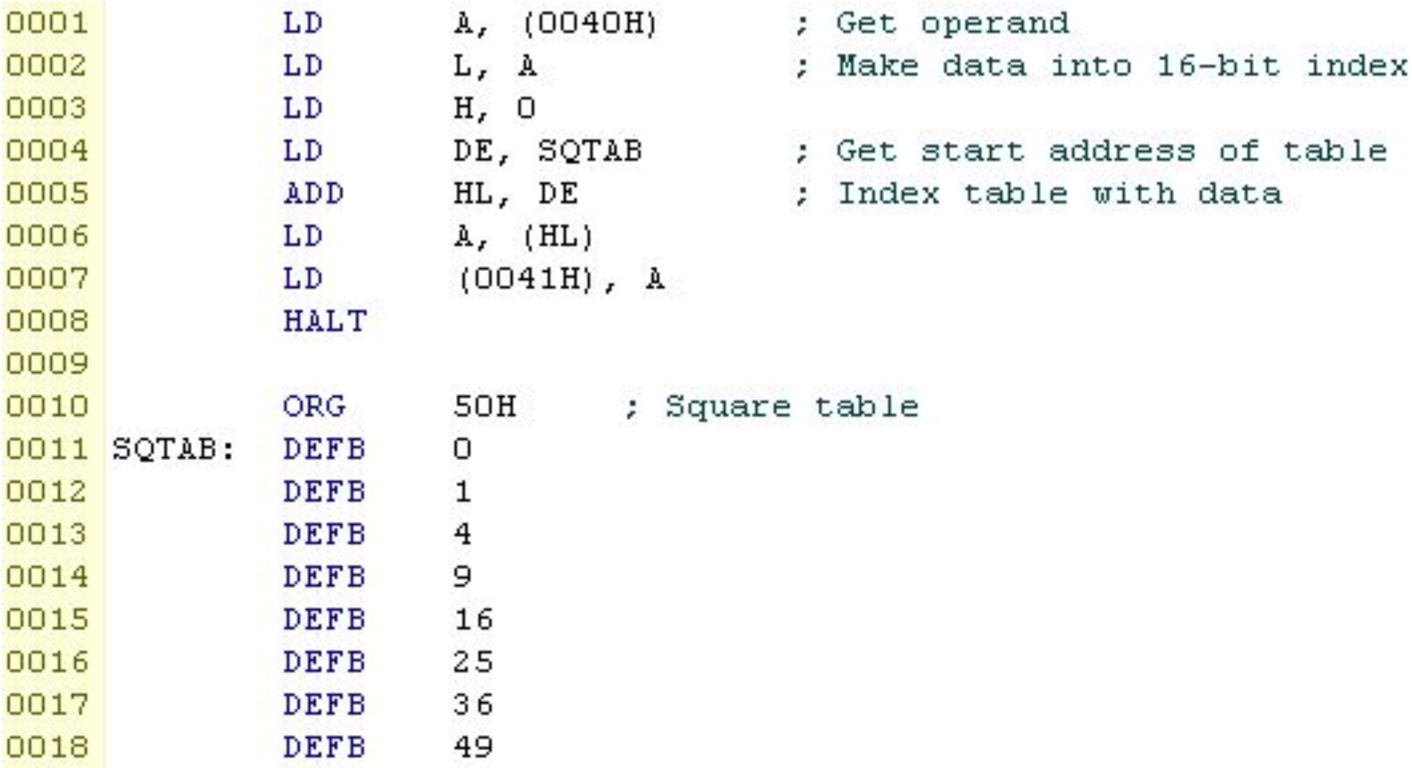
0040	13
	A1
0042	22
	17
0043	

0040	13
	A1
0042	22
	17
0043	35
	B8

A113 + 1722 = B835

Programming patterns: Table lookup

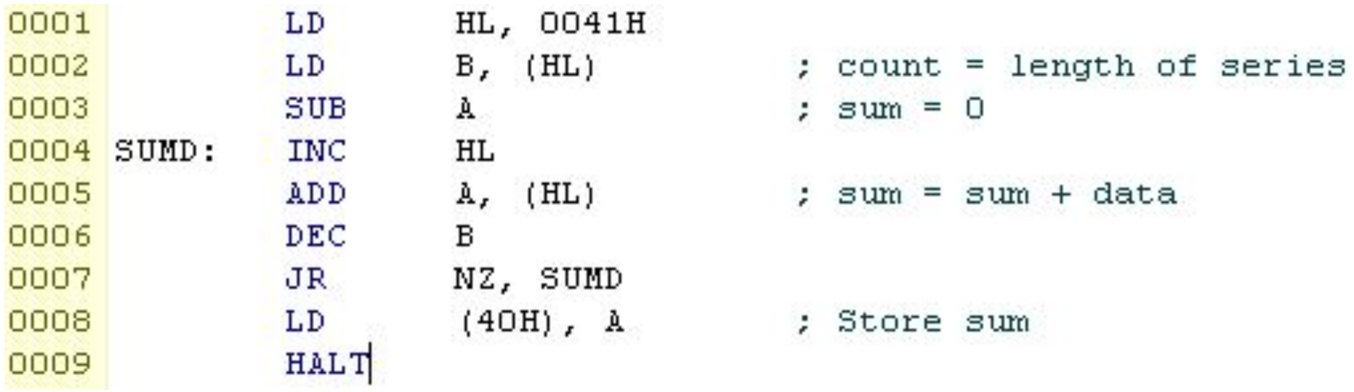
- Example: table lookup (square)
  - Calculate the square of the contents of memory location 0040H from a table and place it into location 0041H. The operand is between 0 and 7
- Solution



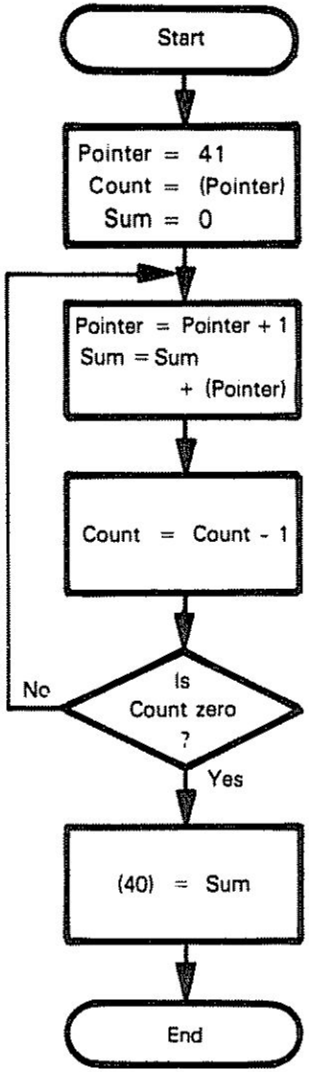
```
        LD      A, (0040H)      ; Get operand
        LD      L, A            ; Make data into 16-bit index
        LD      H, 0
        LD      DE, SQTAB       ; Get start address of table
        ADD     HL, DE          ; Index table with data
        LD      A, (HL)
        LD      (0041H), A
        HALT

        ORG     50H     ; Square table
SQTAB:  DEFB    0
        DEFB    1
        DEFB    4
        DEFB    9
        DEFB    16
        DEFB    25
        DEFB    36
        DEFB    49
```

Programming patterns: Loops

- Example: sum loop
  - Sum a series of numbers. The length of the series is in location 0041H and the series itself begins in location 0042H. The result should be stored in 0040H
- Solution



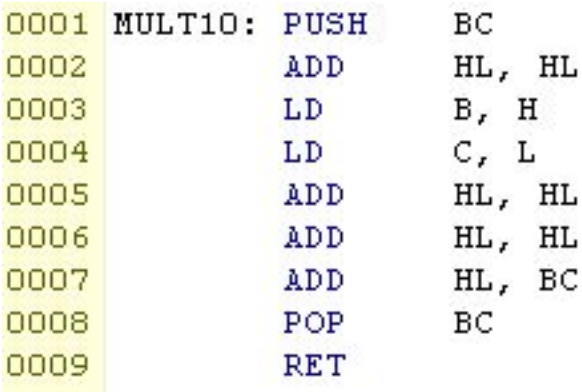
```
         LD     HL, 0041H
         LD     B, (HL)          ; count = length of series
         SUB    A                ; sum = 0
SUMD:    INC    HL
         ADD    A, (HL)          ; sum = sum + data
         DEC    B
         JR     NZ, SUMD
         LD     (40H), A         ; Store sum
         HALT
```



- DJNZ
  - commonly used instruction in loops
  - Let's refer to handbook…

Programming patterns: Subroutines

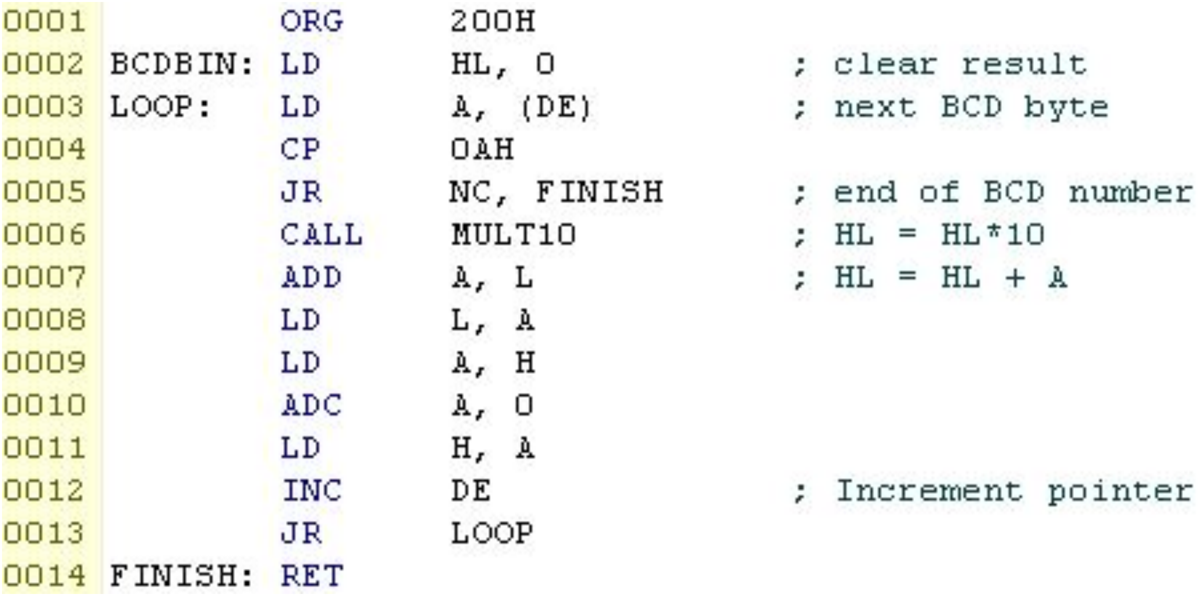
- Example: MULT10 subroutine
  - Write a subroutine that multiplies by 10 a 16-bit number stored in HL
- Solution



```
MULT10: PUSH    BC
        ADD     HL, HL
        LD      B, H
        LD      C, L
        ADD     HL, HL
        ADD     HL, HL
        ADD     HL, BC
        POP     BC
        RET
```

Programming patterns: Fromat conversion

- Purpose: Subroutine to convert BCD to binary
  - The (multi-byte) number to be converted is located in a memory location addressed by DE. The number is terminated by a non-numeric byte. The result should be returned in HL
- Solution



```
        ORG     200H
BCDBIN: LD      HL, 0           ; clear result
LOOP:   LD      A, (DE)         ; next BCD byte
        CP      0AH
        JR      NC, FINISH      ; end of BCD number
        CALL    MULT10          ; HL = HL*10
        ADD     A, L            ; HL = HL + A
        LD      L, A
        LD      A, H
        ADC     A, 0
        LD      H, A
        INC     DE              ; Increment pointer
        JR      LOOP
FINISH: RET
```