Lecture 16 : Software for Micro Processor

3 Levels of Language

1. Machine Language or Machine Code

:Only language that is acceptable to the Microprocessor

But it is not human-readable.

Ex)03C3(=0000 0011 1100 1100 in Binary) is ADD function

for Intel's 80x86.

2. Assembly Language or Low level language

: Language that one to one corresponds to the machine language, and it is human-readable

Ex) ADD AX, BX ; AX <- AX+BX

'Add AX to BX then store the result into AX'

This is one to one corresponding to 03C3. We need compiler (or Assembler) to translate the Assembly language into the Machine Code.

The extension name of source code in Assembly language is .ASM, typically, and it is widely used for low level language programming.

Assembler language gives in-depth understanding for CPU processing and mechatronics devices.

3. High Level Language

: Similar to mathematical expression, thus human-readable

In C++, C, FORTRAN, PASCAL, etc.

Ex) X:=X+Y ; Add X to Y, then store the result into X

The extension names of source code in each high level language are :

.C (for C); .CPP (for C++); .PAS (for Pascal); .FOR (for Fortran); .BAS (for Basic), typically.

This is one of widely used languages, and we need compiler, or translator to translate into the Machine code.

Compiler or Translator

All high level language or Assembler language are to be translated into Machine code (or Machine language) for use in micro-processors.

There are many commercially available compilers and are much depending on the hardware specifications of micro-processors, that indicates proper compiler should be used for specific micro-processor chips.

Once compiling is completed, then the corresponding machine code (or object code) is generated, and the extension name is .OBJ, typically

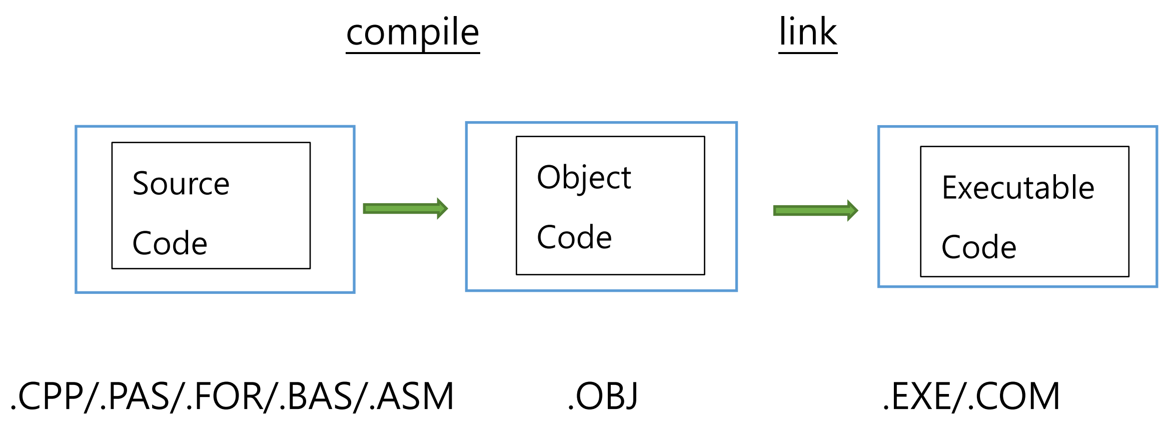
Linker

Although compiling process is completed, the generated machine code cannot be run until the machine code is allocated or positioned to each Segment (such as Code Segment, Data Segment, Extra Data Segment, and Stack Segment) of the memory map of the microprocessor system. This process is called Link, and is performed by a software such as Linker.

During the linking procedures, the several object codes can be linked together with the relevant library codes.

After the linking process, the executable software is generated with extension names such as .EXE or .COM. This executable software is then loaded into microprocessor system, and can be executed.

Generation of Executable Software Codes and extension names in Microprocessor



Assembly Language Programming

Assembly language is a low level language that can drive the micro-processor in the very bottom level, based on the one to one corresponding to the machine code. Thus the Assembly is a very useful tool for in-depth understanding of the microprocessor structure and functioning, although programming itself is not so friendly to users in programming/debugging, etc.

General format

[Label:] Mnemonic DST_operand, SRC_operand ; comments

[ ] symbol indicates optional use

Label=Identifier assigned as the address of the first byte of instruction

Mnemonic=Instruction code such as ADD, MOV

DST_operand, SRC_operand=Destination operands, Source operands that are expressed as the data addressing mode such as Register, Memory, I/O

; Symbol after which comments start, and the comments are not translated

Constants (or Immediate)= Numbers such as

$1011_{(2)}$=1011B, $223_{(10)}$=223 or 223D

$3A_{(16)}$=3AH, $B25A_{(16)}$=0B25AH

Char (or String)='A'=41H

1. Data transfer instruction

MOV DST, SRC ; DST <- SRC

Ex) MOV AX, BX; AX <- BX

MOV [100], DX ; Memory #100 <- DX

MOV AX,[200]; AX <- Memory #200

MOV CX, 100[BP][SI]; CX <- Memory # (100+BP+SI)

Instruction Execution Time=No. of Clock Cycle x Clock Period

Ex) Clock Cycles for MOV

2 for register-register; 4 for register-immediate;

8+EA for register-memory; 9+EA for memory-register

10+EA for memory-immediate

where EA=Effective Address=Displacement(or Distance) within Segment, and it counts up/down by each time pulse

2. Arithmetic Instruction

ADD DST, SRC; DST <- DST+SRC

ADC DST, SRC; DST <- DST+SRC+Carry

SUB DST, SRC; DST <- DST-SRC

```
SBB DST, SRC; DST <- DST-SRC-Carry

CMP DST, SRC; DST-SRC

  (This instruction is just for PSW set without storing the result,

    thus very fast!)

INC OPR; OPR <- OPR+1

DEC OPR; OPR <- OPR-1

NEG OPR; OPR <- OPR

CBW; Extend sign of AL to AH:AL

CWD; Extend sign of AX to DX:AX


IMUL SRC; AX <- AL*SRC if SRC is byte

                ; DX:AX <- AX*SRC if SRC is word

MUL SRC; the same as IMUL except it is unsigned


IDIV SRC;

; AL <- Quotient of AX/SRC if SRC is byte

; AH <- Remainder of AX/SRC if SRC is byte

; AX <- Quotient of DX:AX/SRC if SRC is word

; DX <- remainder of DX:AX/SRC if SRC is word

DIV; The same as DIV except it is unsigned
```

3. Logical Instruction

NOT OPR; OPR <- $\underline{\text{OPR}}$

OR DST SRC; DST <- DST$\vee$SRC

AND DST, SRC; DST <- DST$\wedge$SRC

XOR DST, SRC; DST <- DST XOR SRC

TEST OPR1, OPR2; OPR1$\wedge$OPR2;

(TEST is just for AND operation only without storing the result,

thus very fast!)

4. I/O instruction

MOV DX, address

IN AL, DX; AL <- (DX) if one byte input, where () indicates address

IN AH, DX; AH <- (DX) if one byte input

IN AX, DX; AX <- (DX) if one word input

OUT DX, AL; (DX) <- AL if one byte output

OUT DX, AH; (DX) <- AH if one byte output

OUT DX, AX; (DX) <- AX if one word output

5. Jump (or branch) instruction

JZ label ; Jump to label on Zero(=0)

JNZ label ; Jump to label on Nonzero(≠0)

JB label; Jump to label on Below zero(<0)

JBE label; Jump to label on Below or Equal (≤0)

JG label; Jump to label on Greater than zero (>0)

JGE label; Jump to label on Greater than or Equal to zero (≥0)

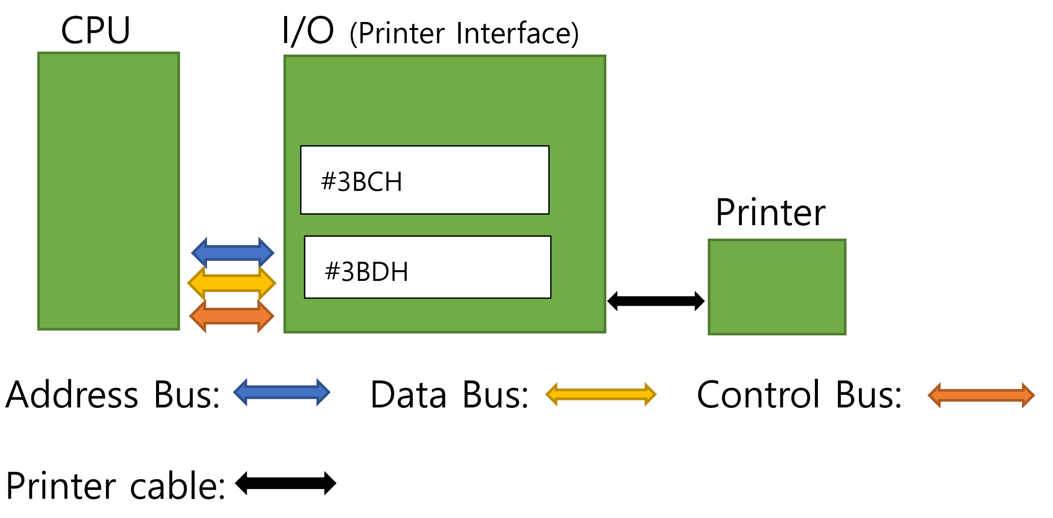
JMP label; Unconditional JUMP to label

Ex) Design a Microprocessor system structure such that

Print 'A' character through a printer, where

Address of Printer Status port=3BDH

Address of Printer Data port=3BCH



<At #3BDH>

Bit pattern=Bit7 Bit6 Bit5 Bit4 Bit3 Bit2 bit1 Bit0

```
                1    1    0    1    1    x    x    x
```

where Bit7=0 if printer is busy, otherwise 1

Bit6=1 if normal input to printer, otherwise 0 (for acknowledge pulse)

Bit5=0 if printer has paper, otherwise 1 (no paper)

Bit4=1 if printer is on-line, otherwise 0

Bit3=1 if printer is normal, otherwise 0

Bit2,Bit1,Bit0= Don't Care Bits, thus to be masked

<At #3BCH>

Bit Pattern=Data (ASCII Code) to be printed

<Assembly Code (or Machine Code)>

```
MOV DX, 3BDH

Loop: IN AL, DX

      AND AL, 11111000B ; masking the 'Don't Care Bits'

      CMP AL,11011000B ; checking the normal status of printer

      JNZ Loop

MOV DX, 3BCH

MOV AH, 41H

OUT DX, AH
```

Ex2) Sample Full Assembly Code

```
DSEG SEGMENT
DSEG ENDS
SSEG SEGMENT
SSEG ENDS
CSEG SEGMENT
ASSUME DS:DSEG, SS:SSEG, CS:CSEG
        MOV AX,7H
        MOV BX,7H
        SUB AX,BX
        ADD AX,BX
        IMUL BL
        IDIV BL
        OR AH, 00001111B
        AND BL, 00001111B
        NOT AX
        MOV DX, 3BDH
LOOP1: IN AL, DX
        AND AL, 11111000B
        CMP AL, 11011000B
        JNZ LOOP1
        MOV DX, 3BCH
        MOV AL, 41H; ‘A’
        OUT DX, AL
CSEG ENDS
END
```
