Microprocessors & Interfacing

AVR Programming (III)

Lecturer : Annie Guo

Lecture Overview

- Memory access
- Assembly process
  - First pass
  - Second pass

Memory Access Operations

- Access to data memory
  - Using instructions
    - ld, lds, st, sts
- Access to program memory
  - Using instructions
    - lpm
    - spm
      - Not covered in this course
  - Most of the time, that we access the program memory is to load data

Load Program Memory Instruction

- Syntax: *lpm Rd, Z*
- Operands: Rd∈{r0, r1, …, r31}
- Operation: Rd ← (Z)

- Words: 1
- Cycles: 3

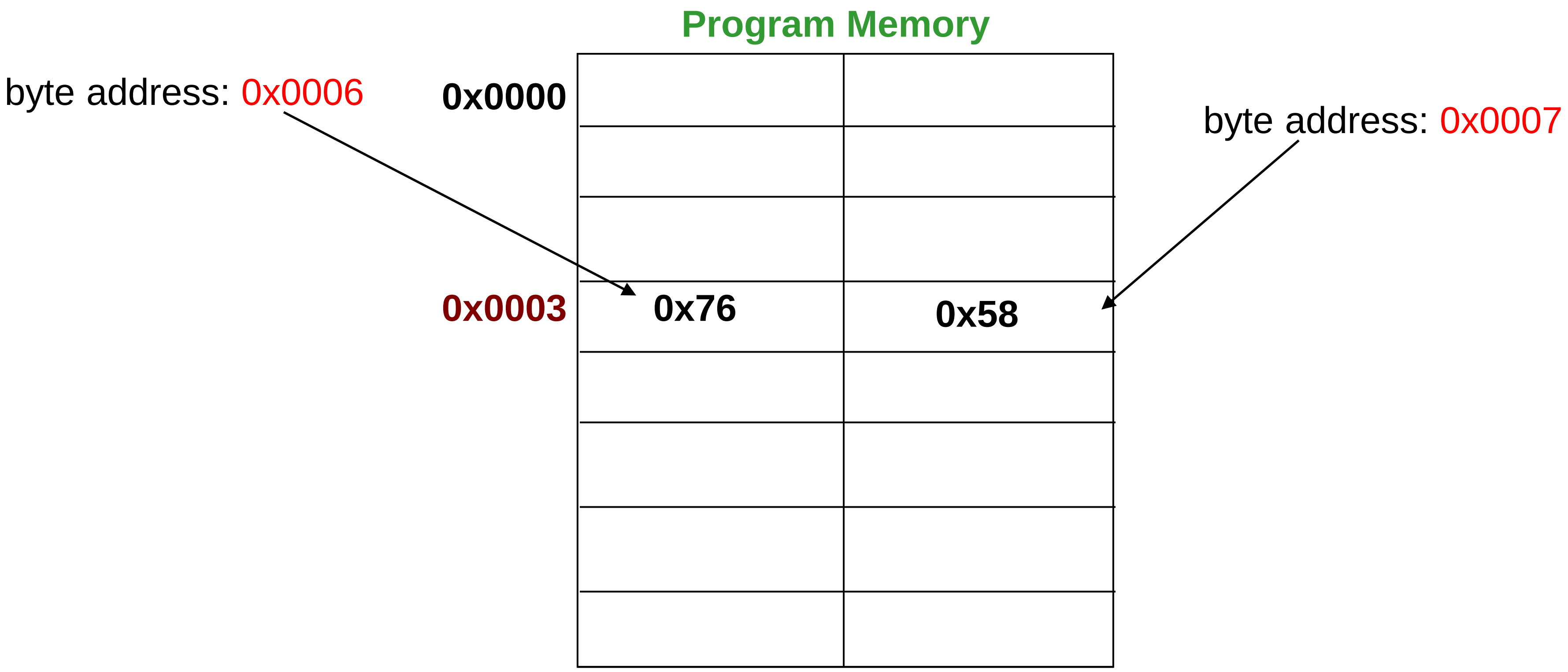
0x0000	‘C’	‘O’
0x0001	‘M’	‘P’
0x0002	‘9’	‘0’
0x0003	‘3’	‘2’
0x0004	0	0
0x0005	0x48	0x23

Load Data From Program Memory

- The address label in the program memory is a ***word address.***
- To access constant data in the program memory with instruction *lpm*, ***byte address*** should be used.
- Address register, Z, is used to point to a byte in the program memory

Byte Address vs Word Address

- First-byte-address (in a word) = 2 * word-address
- Second-byte-address (in a word) = 2 * word-address +1



Program Memory

byte address: 0x0006

0x0000

byte address: 0x0007

0x0003

0x76

0x58

•LSB – least significant bit

Example

```
.include "m2560def.inc"    ; include definition for Z

ldi ZH, high(Table_1<<1)   ; initialize Z
ldi ZL, low(Table_1<<1)      <<1:将word address转化为byte address

lpm r16, Z                 ; load constant from the program
                           ; memory pointed to by Z (r31:r30)
            ⋮

Table_1:
                                          LSB:最低有效位
        .dw 0x5876         ; 0x76 is the value when Z_LSB = 0   means the first byte'76'
                           ; 0x58 is the value when Z_LSB = 1
```

Complete Example 1

- Copy data from Program memory to Data memory

Complete Example 1 (cont.)

- C description

```
struct STUDENT_RECORD
{
            int student_ID;
            char name[20];
            char WAM;
};

typedef struct STUDENT_RECORD student;

student s1 = {123456, “John Smith”, 75};
```

Complete Example 1 (cont.)

- Assembly translation

```
.include   “m2560def.inc”

.set       student_ID=0
.set       name = student_ID+4
.set       WAM = name + 20
.set       STUDENT_RECORD_SIZE = WAM + 1

.cseg
start:     ldi zh, high(s1_value<<1)       ; pointer to student record
           ldi zl, low(s1_value<<1)        ; value in the program memory

           ldi yh, high(s1)                ; pointer to student record holder
           ldi yl, low(s1)                 ; in the data memory

           clr r16
```

Complete Example 1 (cont.)

- Assembly translation (cont.)

```
load:
                cpi r16, STUDENT_RECORD_SIZE
                brge end
                lpm r10, z+         Load Program Memory and Post-Inc
                st  y+, r10          Store Indirect and Post-Inc.
                inc r16
                rjmp load
end:
                rjmp end

s1_value:       .dw     LWRD(123456)
                .dw     HWRD(123456)
                .db     "John Smith          ", 0       ;take 20 bytes
                .db     75
.dseg
.org 0x200
s1:     .byte   STUDENT_RECORD_SIZE
```

Complete Example 2

- Convert lowercase to uppercase for a string (for example, “hello”)
  - The string is stored in the program memory
  - The resulting string after conversion is stored in the data memory.
  - In ASCII, uppercase letter + 32 = lowercase letter
    - e.g. ‘A’+32=‘a’

Complete Example 2 (cont.)

- Assembly program

```
.include "m2560def.inc"
.equ size = 6                                   ; string length
.def counter = r17
.dseg
.org 0x200                                      ; set the starting address
                                                ; of data segment to 0x200
ucase_string:  .byte  size

.cseg
            ldi zl, low(lcase_string<<1)        ; get the low byte for
                                                ; the address of "h"
            ldi zh, high(lcase_string<<1)       ; get the high byte for
                                                ; the address of "h"
            ldi yh, high(ucase_string)
            ldi yl, low(ucase_string)
            clr counter                         ; initialize counter
```

Complete Example 2 (cont.)

- Assembly program (cont.)

```
main:
        lpm  r20, z+     ; load a letter from flash memory
        subi r20, 32     ; convert it to the uppercase letter
        st   y+,r20      ; store the uppercase letter in SRAM
        inc  counter
        cpi  counter, size-1
        brlt main    Branch if Less , Signed
        lpm r20, z       ; copy null
        st y, r20
end:
        rjmp end

lcase_string:  .db   “hello”, 0
```

Assembly

- Assembly programs need to be converted to machine code before execution
  - This translation/conversion from assembly program to machine code is called ***assembly*** and is done by the ***assembler***
- There are two general steps in the assembly processes:
  - Pass one
  - Pass two

Two Passes in Assembly

- Pass One
  - Do lexical and syntax analysis: checking for syntax errors
  - Expand macros
  - Record all the symbols (labels etc) in a symbol table
- Pass Two
  - Use the symbol table to substitute values for symbols and evaluate functions.
  - Assemble each instruction
    - i.e. generate machine code

Example

Assembly program

```
.equ       bound = 5

           clr r16
loop:
           cpi r16, bound
           brlo end
           inc r16
           rjmp loop
end:
           rjmp end
```

Symbol table

Symbol	Value
bound	5
loop	1
end	5

Example (co

Code generation

```
.equ        bound = 5

            clr r16
loop:
            cpi r16, bound
            brlo end
            inc r16
            rjmp loop
end:
            rjmp end
```

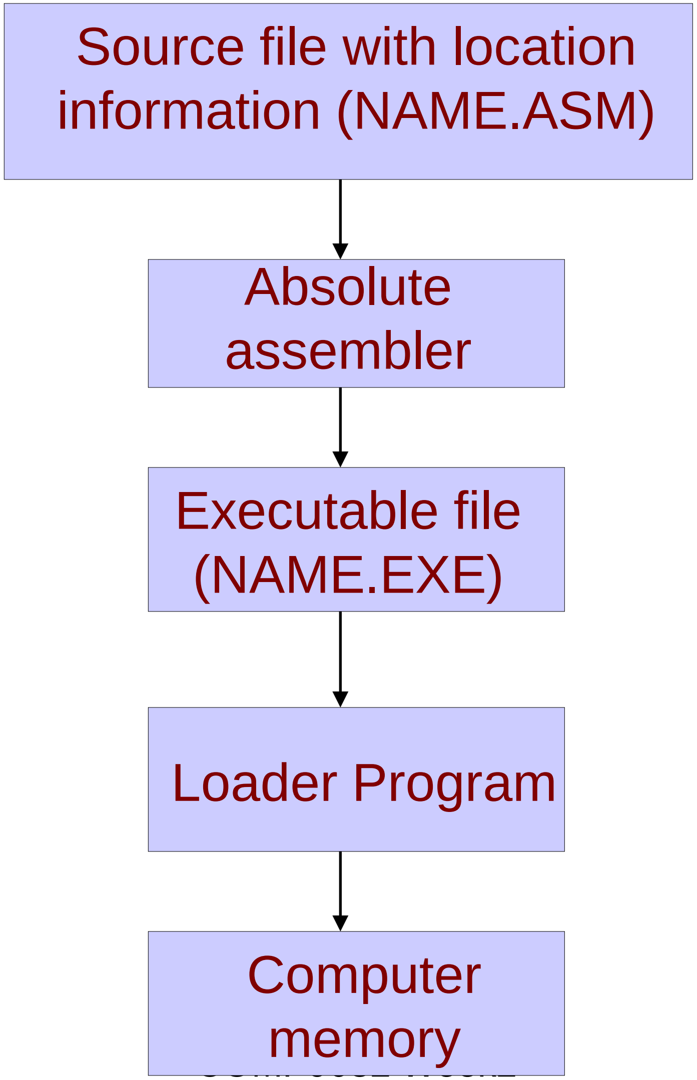
```
Address          Code          As

00000000:        2700          clr     r16
00000001:        3005          cpi     r16,0x05
00000002:        F010          brlo    PC+0x02
00000003:        9503          inc     r16
00000004:        CFFC          rjmp    PC-0x0004
00000005:        CFFF          rjmp    PC-0x0001
```

Absolute Assembly

- A type of assembly process.
  - Can only be used for the source file that contains all the source code of the program
- Programmers use .org to tell the assembler the starting address of a segment (data segment or code segment)
- Whenever any change is made in the source program, all code must be assembled.
- A loader transfers an **executable file** (machine code) to the target system.

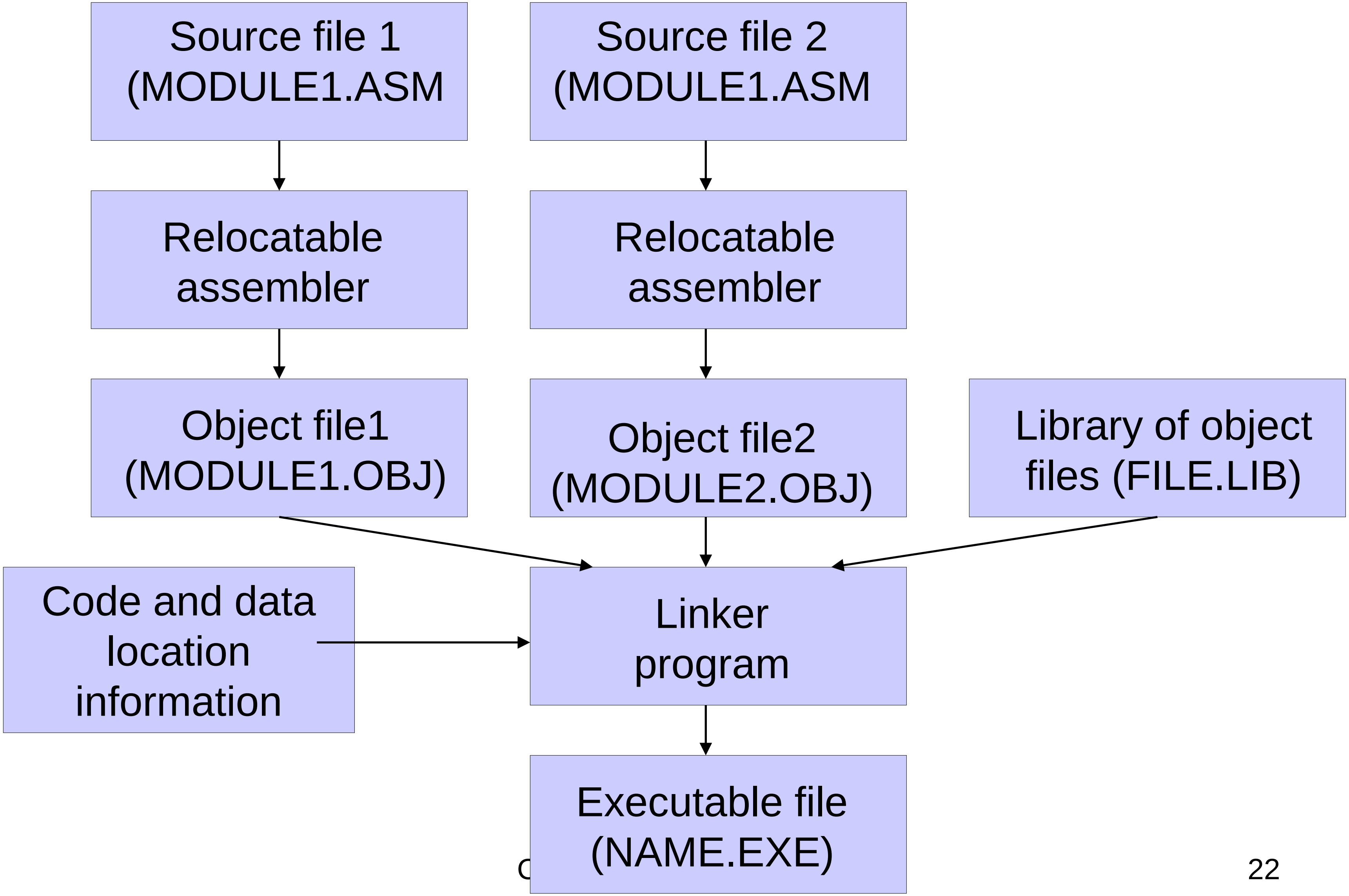
Absolute Assembly - workflow


Source file with location information (NAME.ASM)
Absolute assembler
Executable file (NAME.EXE)
Loader Program
Computer memory

Relocatable Assembly

- Another type of assembly process.
- Each source file can be assembled separately
- Each file is assembled into **an object file** where some addresses may not be resolved
- A linker program is needed to resolve all unresolved addresses and make all object files into a single executable file

Relocatable Assembly - workflow



Source file 1 (MODULE1.ASM

Source file 2 (MODULE1.ASM

Relocatable assembler

Relocatable assembler

Object file1 (MODULE1.OBJ)

Object file2 (MODULE2.OBJ)

Library of object files (FILE.LIB)

Code and data location information

Linker program

Executable file (NAME.EXE)

Homework

1. Write a macro that can perform either logical shift left or arithmetic shift right on a register by a given number of bits.
2. Write a macro to check whether a register holds a valid hexadecimal digit.