



移动信息工程学院

School of Mobile Information Engineering

Computer Architecture

exercise2 Assembler(accounting exercise)

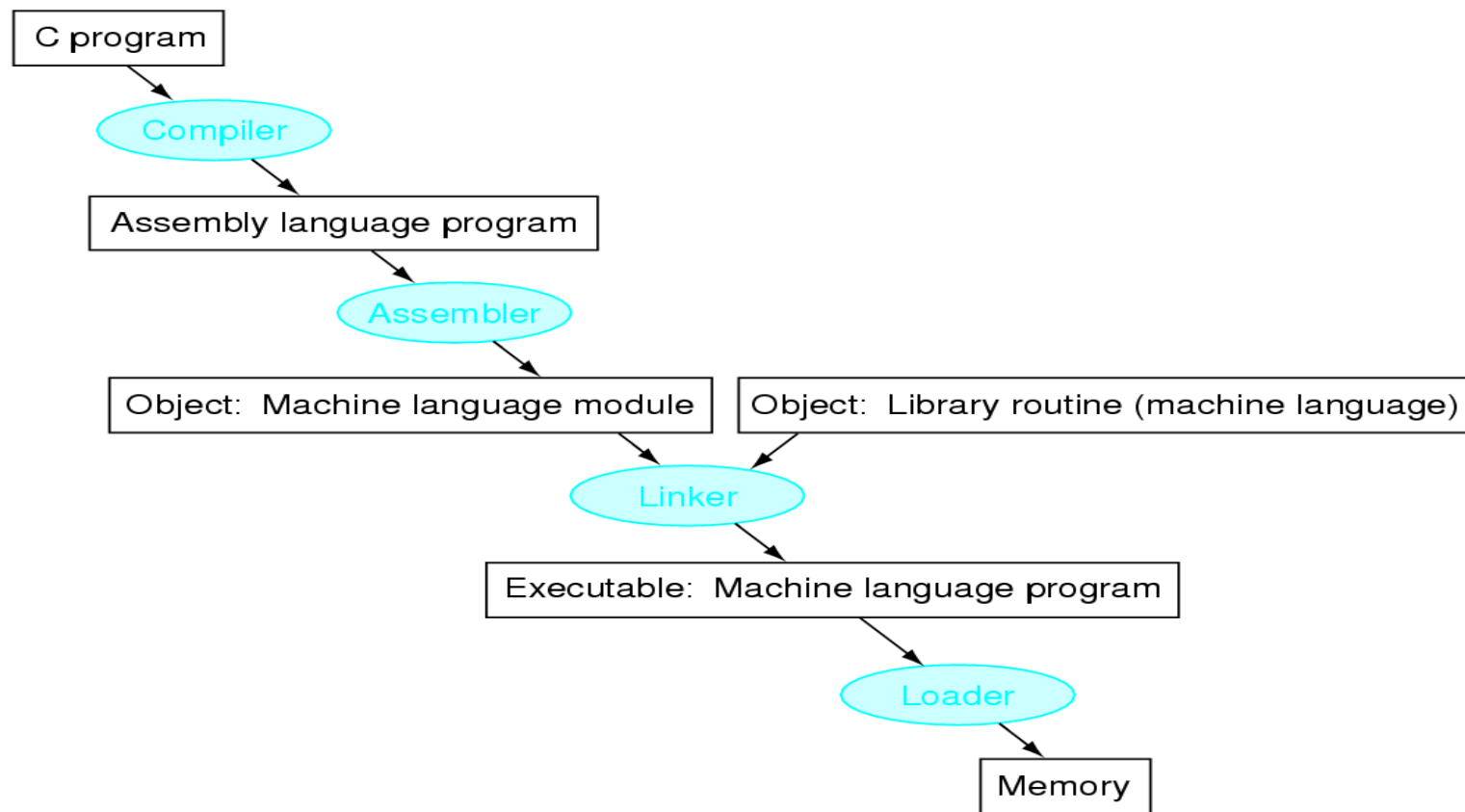
2016/4/16





Requirements

- 将高级语言（**c**）进行编译和反汇编后得到汇编代码
- 对生成的**.s**文件（汇编代码）进行注释





➤ compilation

```
arm-elf-gcc -c matrix.c
```

➤ disassembly

```
arm-elf-objdump -S matrix.o
```



Example: 矩阵相乘

$$\begin{array}{c} \mathbf{A} \qquad \times \qquad \mathbf{B} \qquad =: \qquad \mathbf{C} \\ \\ \left(\begin{array}{ccc} * & \cdots & * \\ \vdots & \diagdown & \vdots \\ * & \cdots & * \\ * & \cdots & * \end{array} \right) \left. \vphantom{\begin{array}{ccc} * & \cdots & * \\ \vdots & \diagdown & \vdots \\ * & \cdots & * \\ * & \cdots & * \end{array}} \right\} \mathbf{n} \times \left(\begin{array}{ccc} * & \cdots & * & * \\ \vdots & \diagdown & \vdots & \vdots \\ * & \cdots & * & * \\ * & \cdots & * & * \end{array} \right) \left. \vphantom{\begin{array}{ccc} * & \cdots & * & * \\ \vdots & \diagdown & \vdots & \vdots \\ * & \cdots & * & * \\ * & \cdots & * & * \end{array}} \right\} \mathbf{m} =: \left(\begin{array}{ccc} * & \cdots & * & * \\ \vdots & \diagdown & \vdots & \vdots \\ * & \cdots & * & * \\ * & \cdots & * & * \end{array} \right) \left. \vphantom{\begin{array}{ccc} * & \cdots & * & * \\ \vdots & \diagdown & \vdots & \vdots \\ * & \cdots & * & * \\ * & \cdots & * & * \end{array}} \right\} \mathbf{m} \\ \underbrace{\hspace{1.5cm}}_{\mathbf{m}} \qquad \underbrace{\hspace{1.5cm}}_{\mathbf{n}} \qquad \underbrace{\hspace{1.5cm}}_{\mathbf{m}} \end{array}$$



Example: 矩阵相乘

```
void matrix(int *A, int *B, int *C, int n, int m)
{
    int i,j,k,sum;
    for(i=0; i < n; i++) {
        for(j = 0; j < n; j++) {
            sum= 0;
            for(k = 0; k < m; k++) {
                sum = sum + (A[i * m + k] * B[k * n + j]);
            }
            C[i * n + j] = sum;
        }
    }
}
```



Example: 矩阵相乘

➤ 汇编过程

```
ChenYuxiao@Master:~/Desktop/1$ arm-elf-gcc -c matrix.c
```

➤ 反汇编

```
ChenYuxiao@Master:~/Desktop/1$ arm-elf-objdump -S matrix.o
```

➤ 直接C代码转.s文件

```
ChenYuxiao@Master:~/Desktop/1$ arm-elf-gcc -S matrix.c
```



Task

➤ 冒泡排序

➤ 数组清零

➤ 二进制



Task 1 冒泡排序



要求

- 用C语言实现冒泡排序
- 对实现的代码进行编译和反汇编
(`arm-elf-gcc -c filename.c`)
(`arm-elf-objdump -S filename.o`)
- 对生成的汇编代码（.s文件）进行注释



C代码

```
void Bubble_Sort(int *num, int n) {  
}
```



Task 2 数组清零



要求

- 将3个不同版本的C代码转化为汇编代码
(`arm-elf-gcc -S filename.c`)
- 对生成的.s文件（汇编代码）注释
- 对比3段代码的汇编代码，分析不同之处，说明哪段代码效率更高，原因？



C代码

```
clear1(int array[], int size) {  
    int i;  
    for (i=0; i<size; i+=1)  
        array[i]=0;  
}
```

```
clear2(int *array, int size) {  
    int *p;  
    for (p=&array[0]; p < &array[size]; p=p+1)  
        *p=0;  
}
```

```
clear3(int *array,int size){  
    int *p;  
    int *tmp = &array[size];  
    for(p = &array[0];p<tmp;p++)  
        *p = 0;  
}
```



Task 3 二进制知识讲解

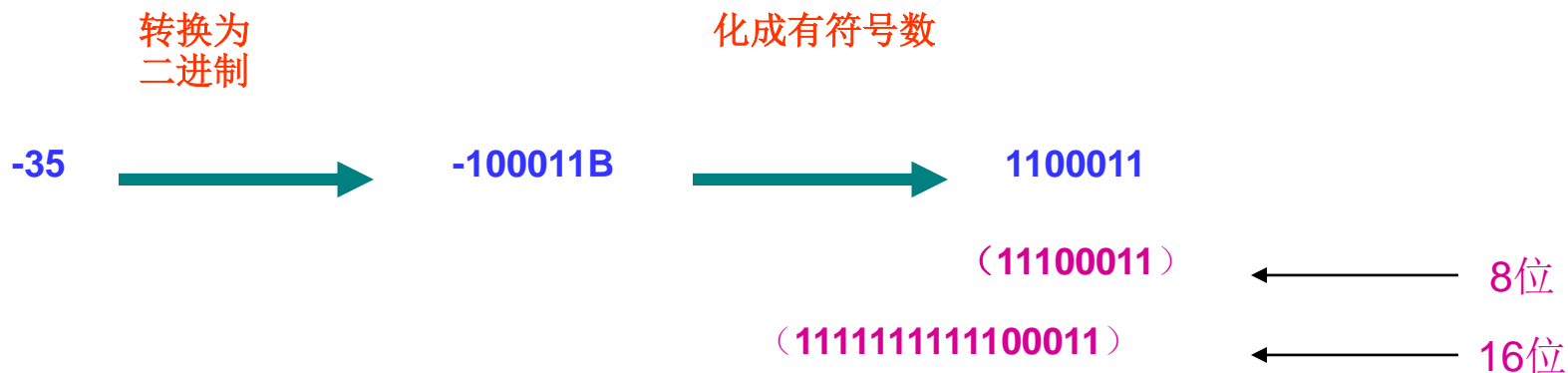
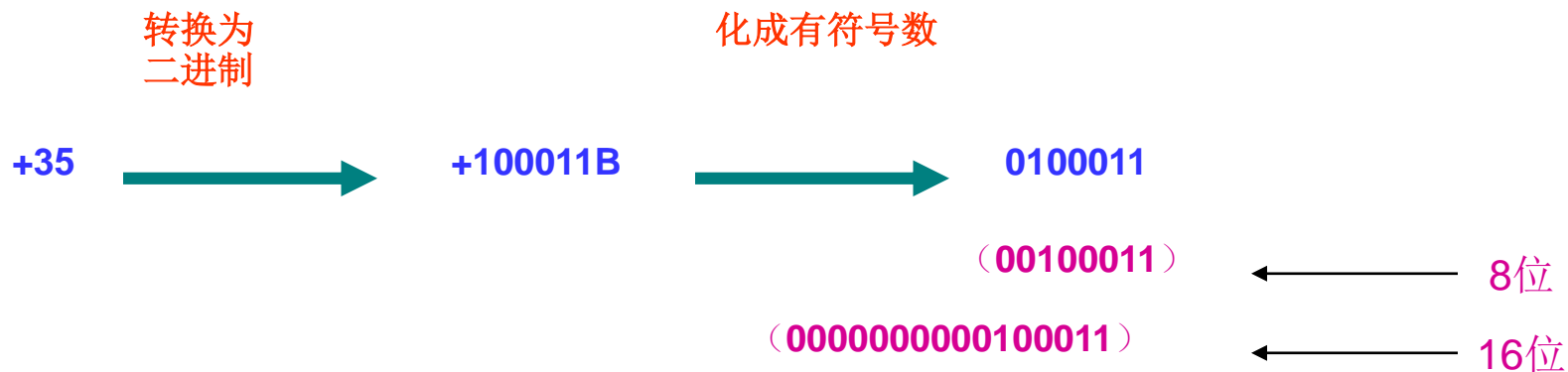


二进制数知识讲解

二进制原码---有、无符号数

- 无符号数；有符号数

— 如： +35





二进制原码---有、无符号数

对于K位的二进制数，数值的大小范围是：

signed: $[-2^{k-1}, 2^{k-1} - 1]$; 例如4位的二进制数, $[-8, 7]$;

unsigned: $[0, 2^k - 1]$; 例如3位的二进制数为 $[000, 111]$, 即 $[0, 7]$

二进制转换成十进制

305.56的按权展开式:

$$3 \times 10^2 + 0 \times 10^1 + 5 \times 10^0 + 5 \times 10^{-1} + 6 \times 10^{-2}$$

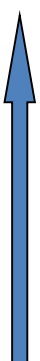
101.01B的按权展开式:

$$1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$$




十进制-二进制转换

- 十进制数的整数部分转换成二进制数可用除法；
- 十进制数的小数部分转换成二进制数可用乘法；
- 例：将26.25转换成二进制数

2	26	余数	
2	13	→ 0	 低位 高位
2	6	→ 1	
2	3	→ 0	
2	1	→ 1	
0		→ 1	

$\therefore 26 = 11010\text{B}$

	0.25	整数	 高位 低位
X	2		
	0.5	0	
X	2		
	1.0	1	

$\therefore 0.25 = 0.01\text{B}$

$\therefore 26.25 = 11010.01\text{B}$



二进制补码---求法

- 正数：与原码相同；
- 负数：求绝对值的二进制数→各位按位取反→ 加1

例：+9 的补码表示方法

$$[+9]_{\text{原码}} = 00001001 = [+9]_{\text{补}}$$

$[-9]_{\text{补}}$

-9绝对值的为： 00001001 + 1

按位取反： 11110110

加1： 11110111

-9的补码就是： 11110111



- 用补码表示计算机中的数后，加减运算均可统一为加法。

例： 设 $x = +0000111$, $y = +0000100$, 计算式子: $x-y$;
(先算出: $[x]_{\text{补}} = 00000111$, $[-y]_{\text{补}} = 11111100$,
再得出: $x-y = x + (-y)$)

补码运算:

$$\begin{array}{r} 00000111 \\ + 11111100 \\ \hline 1 \quad 00000011 \end{array}$$

↑
自然丢失

手工验算:

$$\begin{array}{r} 0000111 \\ - 0000100 \\ \hline 0000011 \end{array}$$





要求 Task1.1

将 1101.0110B 转换成十进制，并写出求解过程；

将 134 转换成二进制数，并写出求解过程；

Task1.2

将 ± 134 转换成补码形式的二进制数，并写出求解过程；



移动信息工程学院

School of Mobile Information Engineering

Thank you !