doi:10.3969/j.issn.1672-4348.2017.03.018

运算器中全加器标志位的逻辑设计与应用

汤龙梅, 陈敏, 许雪林

(福建工程学院 信息科学与工程学院, 福建 福州 350118)

摘要: 介绍全加器实现逻辑基础,给出 4 个常用标志位的生成逻辑,重点分析借位标志和溢出标志的生成逻辑,并给出标志位在 MACH 芯片中的实现过程。最后给出标志位在比较转移等指令中的应用方法。

关键词: 计算机; 组成原理; 全加器; 标志位; 逻辑设计

中图分类号: TP303; G642　　**文献标志码**: A　　**文章编号**: 1672-4348(2017)03-0289-05

Logical design and application of flags for ALU's full-adder

Tang Longmei, Chen Min, Xu Xuelin

(College of Information Science and Engineering, Fujian University of Technology, Fuzhou 350118, China)

Abstract: The logical basis of the implementation of full-adder was described. The generation logic of 4 commonly used flags was presented based on the circuit logic of the full-adder. The generation logic of carrying flag and overflow flag were analysed. Then the implementation of the flags in a MACH chip was focused. Finally, the application of the flags in some instructions such as transfer instruction was discussed.

Keywords: computer; composition principle; full-adder; flag; logical design

运算器的核心部分是对数据信息进行加工处理的算术逻辑运算单元(arithmetic logic unit, ALU),而 ALU 除了给出处理结果,还应给出结果的某些特征,如溢出否、有无进/借位输出等[1]。这些特征以状态标志位的形式提供,是 CPU 程序状态字的一部分。因此,运算器设计乃至 CPU 设计应包含标志位生成电路的设计。随着大规模集成电路制造技术的迅速发展,电子系统设计技术发生了很大的变化,将 EDA 和可编程逻辑器件设计与计算机组成原理授课相结合,让学生将理论应用于实践,已经在很多院校尝试[2-3]。多数计算机组成原理类教材详细介绍了 ALU 中全加器的设计原理[4-7],但对标志位的生成逻辑并没有深入分析。文献[8-10]介绍了 ALU 处理逻辑的实现或优化方法,但没有给出除进位标志以外的标志位的实现过程。事实上,有些标志位如借位标志和溢出标志的生成逻辑并不像其概念那么直接。在全加器/全减器设计基础上,介绍了 4 个常用标志位的生成逻辑,重点分析了借位标志和溢出标志的生成逻辑,并给出了 4 个标志位在 MACH 芯片中的实现过程;最后给出了标志位在一些常用指令中的应用方法。

1　全加器/全减器的设计

基于二进制加法规则设计的全加器,其原理及硬件电路并不复杂。利用全加器实现两个无符号定点数相加,对和以及进位的提取也比较简单。图 1 是 1 位全加器逻辑电路图[1],X_n和 Y_n为数据

收稿日期: 2017-01-07; 修回日期:2017-03-25
基金项目: 福建省教育科学"十二五"规划课题(FJJKCG15-180)
通讯作者: 汤龙梅(1977-),女,江西永新人,讲师,硕士,研究方向:嵌入式技术、机器学习。

输入位，C_{n-1}为低进位输入位，F_n为相加之后的和输出位，C_n为进位输出位。其中，$F_n = X_n \oplus Y_n \oplus C_{n-1}$，$C_n = X_n \cdot Y_n + X_n \cdot C_{n-1} + Y_n \cdot C_{n-1} = \overline{\overline{X_n \cdot Y_n} \cdot \overline{C_{n-1} \cdot (X_n \oplus Y_n)}}$。将多个 1 位全加器级联，可以构建多位全加器。

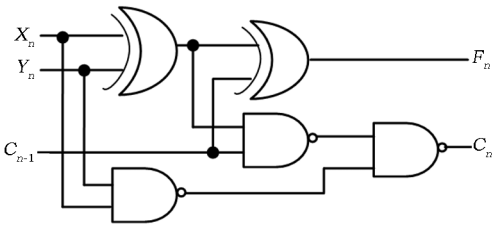

图 1　1 位全加器逻辑电路图

Fig.1　Logic circuit of 1-bit full adder

由于定点数中的小数点并没有被编码成二进制位，因此，小数点对全加器而言是透明的。例如，二进制数 0010b 和 1010b 可表示十进制纯整数 2 和 10，也可表示纯小数 0.125 和 0.625（省略最左边的 0.），将这两个二进制数输入到 4 位全加器上进行相加：0010b + 1010b = 1100b。1100b 既是 12 的二进制形式，也是 0.75 的二进制表示的小数部分（1100b 左边补上 0.即得 0.75），说明同一个全加器既可以实现无符号定点小数（小数点隐藏在最右边）相加，也可以实现无符号定点整数（小数点隐藏在最左边）相加，使用者自己明白小数点的位置即可。

对于有符号定点数，除了数值部分需要被编码，符号也需要被编码为二进制。常用的有符号数编码有原码、反码和补码等。由于采用补码表示时，中符号位可以与数值位一起参与运算，并且可以将减法转换为加法进行运算，因此计算机中一般均采用补码进行加减运算[4-7]。有符号定点小数与有符号定点整数的小数点对加法器而言，仍然是透明的。为避免重复，以下式子或示例均以定点整数为阐述对象。n 位二进制整数补码（含 1 位符号位、$n-1$ 位数值位）加、减法公式分别见式（1）和式（2）[4-7]。

$$[x+y]_{补} = [x]_{补} + [y]_{补} \quad (\mathrm{mod}\ 2^n) \qquad (1)$$

$$[x-y]_{补} = [x]_{补} + [-y]_{补} \quad (\mathrm{mod}\ 2^n)$$
$$= [x]_{补} + [[y]_{补}]_{求补} \quad (\mathrm{mod}\ 2^n) \qquad (2)$$

式（1）说明两个数和的补码等于各自补码相加后的和。在运算结果未超出范围时，对 2^n 求模只需将进位直接舍弃即可得到。仍以上述 0010b +1010b = 1100b 为例，这 3 个数可以分别是+2、−6 和−4 的 4 位补码表示，符合式（1）。当然如果当作定点小数处理，它们也可以是+0.25、−0.75 和 −0.5的 4 位补码表示（省略了最左边的 0.）。

式（2）说明减法运算可转化成加上减数相反数的运算，式中$[[y]_{补}]_{求补}$是将$[y]_{补}$连同符号位在内全部按位取反再加 1。在全加器中增加一组非门将减数取反，并从最低进位位输入 1，就可实现减法运算，因而不需另外设计减法器。Intel 的 74LS181 和 AMD 的 AM2901 内部都是利用全加器和补码来实现减法运算的。

为实现取反操作，在图 1 的基础上增加一个异或门和一个控制信号 M，将原始 Y_n和 M 经该异或门处理后再作为图 1 中的 Y_n。当 $M=0$ 时，Y_n 与 M 异或之后仍然是 Y_n，全加器做加法运算；当 $M=1$ 时，Y_n与 M 异或之后将变为 $\overline{Y_n}$，因而全加器做减法运算。

2　标志位生成逻辑分析

表征数据处理结果的特征很多，其中零标志 ZF、符号标志 SF、进/借位标志 CF、溢出标志 OF 是最常用的 4 个标志位。下面依次对它们的生成逻辑进行分析。为便于描述，假设待运算数据为二进制数串 $X(X=X_nX_{n-1}\cdots X_2X_1)$ 和 $Y(Y=Y_nY_{n-1}\cdots Y_2Y_1)$，运算结果为 $F(F=F_nF_{n-1}\cdots F_2F_1)$，低进位输入为 C_0，高进位输出为 C_n。

2.1　零标志 ZF

ZF 标志是表征运算结果是否为零值。当 $F_nF_{n-1}\cdots F_2F_1$全为零时，ZF 取值 1，否则 ZF 取值 0。可采用式（3）来实现。

$$\mathrm{ZF} = \overline{F_n + F_{n-1} + \cdots + F_2 + F_1} \qquad (3)$$

2.2　符号标志 SF

SF 标志表征运算结果是正数还是负数，结果为正数时，SF 取值 0，否则 SF 取值 1。有符号数运算与无符号数运算使用的是同一个加法器，区别只是把最高位看作符号位而不是数值位，因此 SF 直接取 F_n 即可。若 F_n 为 1 即 SF 为 1，表示结果为负数，反之为非负数。

2.3　进/借位标志 CF

CF 标志用来表示加法运算是否产生进位，或减法运算是否产生借位，只对无符号数运算有意

义。加法的进位标志取全加器的最高进位输出位 C_n 即可。串行进位加法的进位生成逻辑比较简单，此处主要讨论借位生成逻辑。

由于减法运算是借助加法来实现的，如果直接采用全加器的进位输出作为借位标志，则与借位定义正好相反。这一点在 AM2901 和 74LS181 上都可以得到体现，下面对此进行分析。

根据 $[[y]_{补}]_{求补}$ 的运算规则，将 $[y]_{补}$ 按位取反等效于 $(2^n-1)-[y]_{补}$，根据式(2)，可得：

$$[x-y]_{补}=[x]_{补}+(2^n-1)-[y]_{补}+1=[x]_{补}-[y]_{补}+2^n.$$

由于此时的 x 和 y 都是无符号数，故：

若 $x\geqslant y$，则 $[x]_{补}\geqslant[y]_{补}$，故 $[x]_{补}-[y]_{补}+2^n\geqslant 2^n$，因而产生进位；

若 $x<y$，则 $[x]_{补}<[y]_{补}$，故 $[x]_{补}-[y]_{补}+2^n<2^n$，因而不会产生进位。

分别以减法 2-3=-1 和 3-2=1 予以例证。为简便起见，补码采用 4 位二进制(下同)。运算器中 2-3 的运算过程是：2 的补码 0010b 和 3 的补码 0011b 分别输入全加器，0011b 按位取反得 1100b，最低进位位送入 1，最后三者相加：0010b+1100b+1 =1111b。1111b 正好是-1 的补码，说明运算结果正确，但 $C_n=0$，与实际有借位刚好相反。

3-2 的运算过程是：3 的补码 0011b 和 2 的补码 0010b 送入全加器，0010b 按位取反得 1101b，最后 0011b+1101b+1=10001。0001b 正好是 1 的补码，但 $C_n=1$，与实际无借位刚好相反。

根据上式分析，生成符合一般意义上的进/借位标志的生成逻辑见式(4)，其中 ADD 和 SUB 分别表示加法和减法运算。

$$\mathrm{CF}=\mathrm{ADD}\cdot C_n+\mathrm{SUB}\cdot\overline{C_n}\qquad(4)$$

文献[1]基于 AM2901 运算器芯片设计了 16 位运算器，将最高位 AM2901 的进位输出 C_{n+4} 直接作为 CF，因此后续将 CF 用作借位标志时要作相应处理。

2.4　溢出标志 OF

OF 标志用于表示当两个有符号数进行加、减运算时，运算结果是否超出了范围。若超出，则 OF 取值 1，否则取值 0。常用的溢出判断方法有 3 种：双进位判断法、单符号位判断法和双符号位判断法[11]。这些方法在某些场合并不完全等价，但相关文献并未给出它们的区别。下面予以对比分析。

2.4.1　双进位判断法

将最高进位 C_n 与次高进位 C_{n-1} 异或得到 OF。双进位法对于加法溢出的判断没有问题，但对借助加法实现的减法溢出判断而言，直接将 $C_n\oplus C_{n-1}$ 作为 OF 输出是有局限性的。以 4 位二进制补码实现-1-(-8)=7 和+1-(-8)=9 为例，-1-(-8)的运算过程是：-1 的补码 1111b 和-8 的补码 1000b 送入运算器，1000b 按位取反得 0111b，低进位位送入 1，最后 1111b+0111b+1 =10111b。0111b 正好为+7 的补码，而次高进位为 0，最高进位为 1，故 OF=1，与实际无溢出相反。类似的，可得到+1-(-8)的运算结果为 1001b，为-7 的补码，显然结果溢出了，但此时 OF 却为 0。

之所以出现上述情况仍然是由于对第二操作数取反导致的。4 位二进制补码数的范围为[-8，+7]，-8 的相反数为+8，已经超出范围了。上例中减去(-8)运算实际变为加上(-8)的运算，导致最后溢出标志出错。因此，当减数为 -2^{n-1} 时，需对原溢出标志取反，增加了处理电路的复杂性，见式(5)。

$$\mathrm{OF}=\mathrm{ADD}\cdot(C_n\oplus C_{n-1})+\mathrm{SUB}\cdot[(Y_n\cdot\overline{Y_{n-1}}\cdot\cdots\cdot\overline{Y_2}\cdot\overline{Y_1})\oplus(C_n\oplus C_{n-1})]\qquad(5)$$

2.4.2　单符号判断法

若两操作数的符号相同，而结果符号与操作数符号相反，表示结果超出范围。可以理解为正数与正数相加，结果得负数；或负数与负数相加，结果得正数。这两种情况都是不可能的，故出现了溢出。

对减法运算溢出判断可以描述为：若两操作数符号相异，而结果符号与减数符号相同(与被减数符号相反)，则溢出。可以理解为正数减负数，结果为负数；或负数减正数，结果为正数。这两种情况也都是不可能的。

综合两种情况，OF 的生成逻辑见式(6)。

$$\mathrm{OF}=\mathrm{ADD}\cdot(X_n\cdot Y_n\cdot\overline{F_n}+\overline{X_n}\cdot\overline{Y_n}\cdot F_n)+\mathrm{SUB}\cdot(X_n\cdot\overline{Y_n}\cdot\overline{F_n}+\overline{X_n}\cdot Y_n\cdot F_n)\qquad(6)$$

该方法依据取反前的数据符号以及结果符号进行判断，对第二操作数取反后是否溢出不影响最终的 OF，因此不会出现方式(1)中的特殊情

况,但仍然比较复杂。

2.4.3 双符号判断法

操作数和运算结果均采用双符号,如果运算结果的两个符号位相异,则说明出现了溢出,因此只需将双符号进行异或即可生成 OF。如上例,-1-(-8)的运算过程是:-1 的补码 11111b 和-8 的补码 11000b 送入运算器,11000b 按位取反得 00111b,最后执行:11111b+00111b+1=100111b,双符号位为 00,故没有溢出。而 1-(-8)的运算过程为:00001b + 00111b+1=01001,结果溢出,此时双符号位 01 中的 0 表示实际的符号,1 是结果的最高数据位,因而正确的结果是+9。

在本方法中,全加器需要额外增加一位,即 n 位二进制补码数相加减需要 $n+1$ 位全加器。虽然增加了硬件开销,但简化了 OF 的生成逻辑,且容易获取正确的运算结果。要注意的是,双符号由 ALU 内部产生用于溢出判断,对用户来说是透明的,其他标志位仍按原有逻辑。

3 标志位的实现

笔者用计算机组成原理实验箱 TEC-2000A 上的 iM4A5-128/64 芯片作为可编程逻辑器件,采用 LATTICE 公司的 ispLEVEL 开发环境和 ABEL 语言,设计了一个包含 4 个标志位的 4 位全加器。利用实验板上的拨动开关输入 $X1 \sim X4$ 和 $Y1 \sim Y4$、低进位 C_0 以及运算功能选择 M(输入 0 给 M 表示做加法,输入 1 给 M 表示做减法),将运算结果 $F1 \sim F5$ 和各状态位送往 LED 指示灯进行显示,测试结果符合上述分析。主要语句如下:

```
EQUATIONS
//5-bit 串行进位 ALU, M = 0 加法, M = 1 减法
    F1 = X1 $ (Y1 $ M) $ C0;
    C1 =(X1&(Y1 $ M)) # (X1&C0) # ((Y1 $ M)&C0);
    F2 = X2 $ (Y2 $ M) $ C1;
    C2 =(X2&(Y2 $ M)) # (X2&C1) # ((Y2 $ M)&C1);
    F3 = X3 $ (Y3 $ M) $ C2;
    C3 =(X3&(Y3 $ M)) # (X3&C2) # ((Y3 $ M)&C2);
    F4 = X4 $ (Y4 $ M) $ C3;
    C4 =(X4&(Y4 $ M)) # (X4&C3) # ((Y4 $ M)&C3);
    F5 = X4 $ (Y4 $ M) $ C4; //双符号位高位,把 X4 和 Y4 当成 X5 和 Y5;
    C5 =(X4&(Y4 $ M)) # (X4&C4) # ((Y4 $ M)&C4);
    ZF = ! (F1#F2#F3#F4); //零标志
    SF = F4; //符号标志
    CF =(! M &C4) # (M&! C4); //进位/借位标志
    //OF1~OF3 均为溢出标志
    OF1 = (! M&(C4 $ C3))
        #(M&((Y = = [1,0,0,0]) $ (C4 $ C3)));
    OF2 = ! M&(X4&Y4&! F4 # ! X4&! Y4&F4)
    #M&(X4&! Y4&! F4 # ! X4&Y4&F4);
    OF3 = F5 $ F4。
```

在上述代码中,$、#、& 和 ! 分别是异或、或、与和非这 4 种常见逻辑运算。为对比不同溢出标志生产方式,此处实现了双符号位运算,因此,4 位二进制的加减运算需要 5 位全加器,$F1 \sim F4$ 为相加或相减后对应的和或差的输出。ZF 为零标志,从它的逻辑表达式可以看出,只有当 $F1 \sim F4$ 全为 0,ZF 才会输出 1。SF 为运算结果的符号位,直接取自 $F4$。CF 为加法进位输出或者减法的借位输出,当执行减法操作时,需要对 $C4$ 取反才是正确的借位标志值。OF1、OF2、OF3 分别是双进位溢出判断法、单符号溢出判断法和双符号溢出判断法。从逻辑表达式上看,OF3 表达式最简单。

4 标志位的应用

用户通过计算机指令来获取或使用标志位所代表的含义,而指令的执行要依靠 CPU,因此 CPU 设计者需要根据指令功能正确选择标志位来控制指令的执行。与标志位相关的指令数目众多,此处选取了 Intel8086 汇编指令系统中比较容易混淆的带进位减法和比较转移类指令[12]来分析如何应用上述 4 个标志位控制指令的执行,表 1 是这些指令应用标志位情况的汇总说明。当然,本研究的控制方法是实验性的,与 Intel8086CPU 的实现方法没有直接关联。

表 1　常用转移指令标志位取值要求

Tab.1　Value requirement of flags for commonly used transfer instruction

指令格式	指令功能	标志位
SBB Dst,Src	Dest←Dest−Src−CF	CF 取反送低进位
JA/JNBE Dst	无符号数比较大于转移	CF=0 且 ZF=0
JAE/JNB Dst	无符号数比较大于或等于转移	CF=0
JB/JNAE Dst	无符号数比较小于转移	CF=1
JBE/JNA Dst	无符号数比较小于或等于转移	CF=1 或 ZF=1
JG/JNLE Dst	有符号数比较大于转移	SF=OF 且 ZF=0
JGE/JNL Dst	有符号数比较大于或等于转移	SF=OF
JL/JNGE Dst	有符号数小于转移	SF≠OF
JLE/JNG Dst	有符号数小于或等于转移	SF≠OF 或 ZF=1

4.1　带借位减法指令 SBB Dest,Src

该指令的功能是 Dest←Dest−Src−CF，CF 为当前借位标志值。不带借位减法运算由 ALU 内部将减数取反、低进位位补 1 后再执行全加运算。若低进位补 0，则相当于 CF 为时的带借位减法操作；而若低进位补 1，则相当于 CF 为 0 时的带借位减法操作。故 CPU 在控制 SBB 指令执行时，需将借位标志 CF 取反，再送往全加器最低进位。

4.2　无符号数比较判断指令

根据指令功能，无符号比较的 4 类指令可以分为条件正好相反的两组：JA/JNBE 与 JNA/JBE 为一组，JB/JNAE 与 JNB/JA 为一组，因此，确定组内一方的判断条件后，另一方判断条件只需将对方条件取反即可。

若 CF=1，表示两数相减产生了借位，说明被减数小于减数，对应 JB 和 JNAE 的转移条件。反之，若 CF=0，表示没有借位，说明被减数不小于减数，对应 JNB/JAE 的转移条件。

若 CF=0 且 ZF=0，说明被减数不小于减数且两数之差非零，故符合 JA/JNBE 的判断要求。反之，若 CF=1 或 ZF=1 即为 JNA/JBE 的判断要求。

根据指令含义，也有将 JAE 指令的判断条件设为 CF=0 或 ZF=1，将 JB 的判断条件设为 CF=1 且 ZF=0。实际上，满足 ZF=1 的情形是满足 CF=0 情形的子集，相应的若 CF=1，则 ZF 不可能为 1。因此这两条指令的判断条件只需根据 CF 的取值即可。

要注意的是，由于无符号数中的最高位是数值位而不是符号位，故符号标志 SF 不宜用于无符号比较转移指令的转移判断依据。例如 1100b 和 0000b（分别对应无符号十进制数 12 与 0）相减后得 1100b，这种情形下 SF=1，CF=0。很明显，CF=0 表示 12 减 1 不需借位，故能正确判断两数谁大谁小。若根据 SF=1 判断差为负数，进而说明不够减，显然是错误的。

4.3　有符号数比较判断指令

对于有符号数比较，需分溢出和不溢出两情况。若没有溢出，则 SF 为正确的结果符号；若有溢出，虽然运算结果不正确，但大小关系仍然可以判断。此时，SF 取值与实际符号正好相反。与无符号数比较指令分组类似，指令 JG/JNLE 和 JNG/JLE 是一组，指令 JL/JNGE 和 JNL/JGE 是另一组。

若 OF=0 且 SF=0，或者 OF=1 且 SF=1，都表示被减数不小于减数，即 SF=OF 对应指令 JNL 和 JGE 的转移条件。相应的 OF≠SF 对应指令 JL 和 JNGE 的转移条件。

若 OF=0 且 SF=0 且 ZF=0，或者 OF=1 且 SF=1 且 ZF=0。表示被减数大于减数，即 SF=OF 且 ZF=0 对应 JG 和 JNLE 的转移条件。相应的 OF≠SF 或 ZF=1 对应 JNG 和 JNLE 的转移条件。

5　结论

标志位与运算器、指令系统、CPU 以及汇编程序设计关系密切。

（下转第 300 页）

and the Wisdom of Crowds. Citeseer,2010,104(45):17599-17601.
[9] Yang Z,Guo J,Cai K, et al. Understanding retweeting behaviors in social networks[C]//Proceedings of the 19th ACM International Conference on Information and Knowledge Management. Oct 26-30,2010, Toronto,ON,Canada. New York: ACM,2010:1633-1636.
[10] Liben-Nowell D, Kleinberg J. Tracing information flow on a global scale using Internet chain-letter data[J].Proceedings of the National Academy of Sciences,2008,105(12):4633-4638.
[11] Fan P, Li P, Jiang Z, et al. Measurement and analysis of topology and information propagation on Sina-Microblog[C]// 2011 IEEE International Conference on Intelligence and Security Informatics(ISI), July 9-12,2011, Beijing China. Washington:IEEE,2011:396-401.
[12] Webberley W,Allen S,Whitaker R.Retweeting: A study of message-forwarding in twitter[C]//2011 Workshop on Mobile and Online Social Networks (MOSN), Sept 8,2011, Milan,Italy. Washington:IEEE,2011:13-18.
[13] 谢婧,刘功申,苏波,等.社交网络中的用户转发行为预测[J].上海交通大学学报,2013,47(4):585-588.
[14] 匡冲,刘知远,孙茂松.微博转发者的个性化排序[J].山东大学学报(理学版),2014,49(11):31-36.
[15] 严玉良,董一鸿,何贤芒,等.FSMBUS:一种基于 Spark 的大规模频繁子图挖掘算法[J].计算机研究与发展,2015,52(8):1768-1783.
[16] 丁圣勇,闵世武,樊勇兵.基于 Spark 平台的 NetFlow 流量分析系统[J].电信科学,2014,30(10):48-51.
[17] 牛海玲,鲁慧民,刘振杰.基于 Spark 的 Apriori 算法的改进[J].东北师大学报(自然科学版),2016,48(1):84-89.
[18] 徐鹏,林森.基于 C4.5 决策树的流量分类方法[J].软件学报,2009,20(10):2692-2704.
[19] 韩家炜,坎伯.数据挖掘:概念与技术[M].北京:机械工业出版社,2012:213-222.
[20] 陈羽中,郭松荣,陈宏,等.基于并行分类算法的电力客户欠费预警[J].计算机应用,2016,36(6):1757-1761.

(特约编辑:黄家瑜)

(上接第 293 页)

通过对常用标志位 ZF、SF、CF 和 OF 生成逻辑的分析、实现及应用的详细阐述,可以更好地了解运算器的工作原理及设计方法,进一步理解运算器、指令系统以及控制器 3 者之间的联系,提高汇编程序设计能力。

参考文献:

[1] 王诚,宋佳兴.计算机组成与结构[M].2 版.北京:清华大学出版社,2011.
[2] 秦磊华,王小兰,管军.EDA 仿真在组成原理设计性实验中的应用[J].实验室研究与探索,2009,28(4):79-82.
[3] 张磊,郑榕,张军峰.计算机组成原理理论实验教学无缝结合的新方法[J].实验室研究与探索,2013,32(5):168-171.
[4] 白中英.计算机组成与系统结构:第五版(立体化教材)[M].北京:科学出版社,2011.
[5] 裘雪红,李伯成.计算机组成与体系结构[M].北京:高等教育出版社,2009.
[6] 唐朔飞.计算机组成原理[M].2 版. 北京:高等教育出版社,2010.
[7] 王爱英.计算机组成与结构[M].5 版. 北京:清华大学出版社,2013.
[8] 袁波,李树荣,姚素英.一种 8 位单片机中 ALU 的改进设计[J].微电子学与计算机,2006,23(4):71-74.
[9] 朱一杰,张曦,俞军.算术逻辑单元的优化设计[J].微电子学与计算机,2004,21(9):155-157.
[10] 宋文强.12 位 RISC 计算器设计[D].成都:电子科技大学,2011.
[11] 齐广玉,邹建伟.计算机组成原理[M].北京:兵器工业出版社,1997.
[12] 沈美明.IBM PC 汇编语程序设计[M].北京:清华大学出版社,1996.

(特约编辑:黄家瑜)