

模拟器和汇编程序(Simulator and Assembler)

本题目在 **P157** 基础上进行了修改和完善，将简单计算机的指令由 16 条扩充到 32 条，另外设计了两条伪指令，同时增加了四个专用寄存器：CS、DS、SS 和 ES。为了避免重复，本文只重点介绍了扩充部分的内容，因此，对任务要求的完整理解需要参阅原题及编程思路（《C 语言实验与课程设计》附录 8，p245，和第四章 4.4 节，p157）。

1 简单计算机的结构模型

模拟器所模拟的简单计算机其结构模型可以用图 1 来表示，整体结构分为三块：处理器（Processor）、存储器（Memory）和端口（Port）。

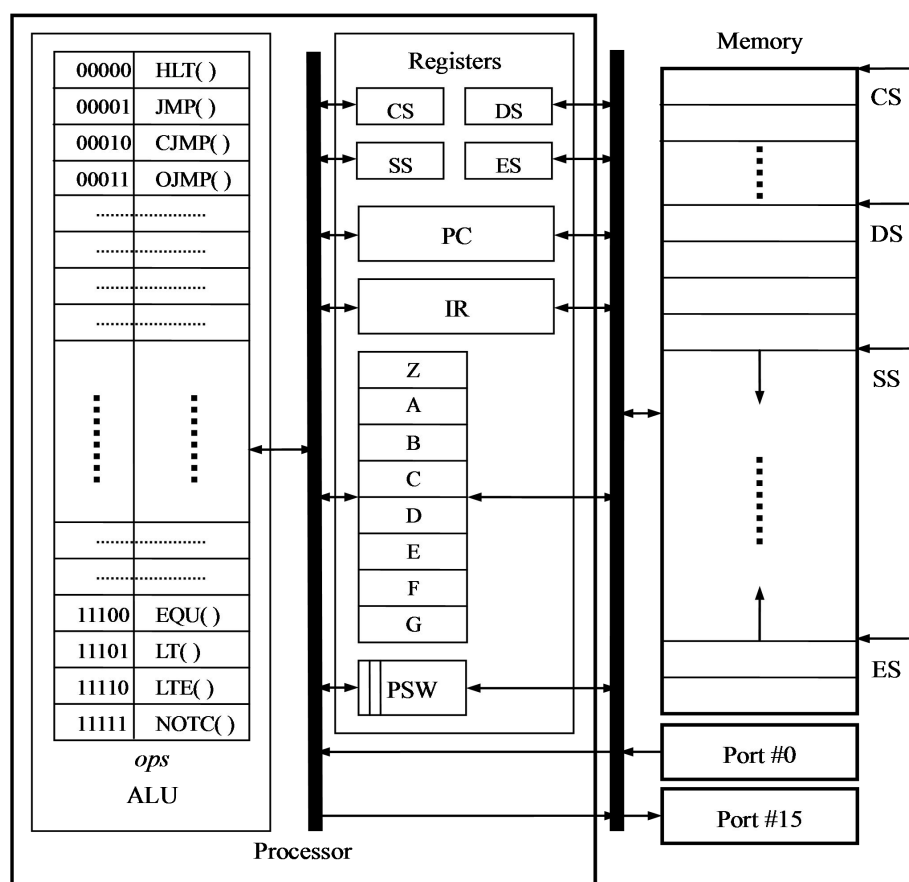


图 1 简单计算机的结构模型

1) 处理器

处理器中包括一个算术与逻辑单元（Arithmetic and Logic Unit, ALU）和一系列寄存器（Registers）。寄存器包括一组通用寄存器（general purpose registers）和七个专用寄存器：代码段寄存器（Code Segment, CS）、数据段寄存器（Data Segment, DS）、堆栈段寄存器（Stack Segment, SS）、附加段寄存器（Extra Segment, ES）、程序计数器（program counter, PC）、指令寄存器（instruction register, IR）和程序状态字（Program Status Word, PSW）。

算术与逻辑单元是处理器的核心部分，用于执行大部分指令。在此计算机中，算术与逻辑单元实际模拟了 32 条指令的执行。

寄存器是在处理器中构建的一种特殊存储器，读写速度高但数量有限。寄存器主要用于存放算术与逻辑单元执行指令所需要的数据和运算结果。处理器所处理的数据大多数取自于寄存器，而不是内存；因此，内存中的数据首先要加载（load）到寄存器，接下来处理器从

寄存器取数据进行运算，然后将运算结果放回寄存器，最后，再存储到内存中。

通用寄存器有 8 个，分别记为 A、B、C、D、E、F、G 和 Z。其中寄存器 Z 用作零寄存器，所存放的值保持为 0，用来“清洗”其他寄存器（即将其他寄存器中的值置为 0）。寄存器 A~G 可存放任何数据，其中寄存器 G 还用于间接寻址，在后面描述指令功能时会进一步介绍。在一条指令在执行时，对同一个寄存器既可以读数据又可以写数据。这 8 个通用寄存器的编号见表 1。

表 1 通用寄存器编号表	
寄存器	编号（二进制）
Z	000
A	001
B	010
C	011
D	100
E	101
F	110
G	111

本简单计算机对内存进行分段管理，内存被划分为四个段，分别为代码段、数据段、堆栈段和附加段，用 CS、DS、SS 和 ES 来表示。代码段存放程序执行指令，数据段存放程序运行所需要的数据，堆栈段存放程序运行期间产生的中间数据，包括子程序调用时所传递的参数和子程序所返回的结果，附加段用于在调用子程序时存放寄存器中的数据。SS 和 ES 被设计成栈式存储结构。

目标程序在执行之前必须先载入到内存。程序计数器专门用来存放内存中将要执行的指令地址。由于每条指令占 4 字节，因此一般情况下每经过一个指令周期，程序计数器中的地址自动向前移动 4 字节，而转移指令可以修改程序计数器中的地址值，从而改变指令的执行次序。

处理器的每个指令周期包括三步：取指、译码和执行。指令寄存器专门用来存放从程序计数器所指示的内存单元取到的指令内容，以供下一步对其译码。

程序状态字专门存放目标程序执行的状态。该简单计算机用两个标志来表示程序执行时两个方面的状态：溢出标志（overflow flag）用来表示算术运算（ADD 和 SUB 等）时的溢出状态，1 表示发生溢出，0 表示未溢出；比较标志（compare flag）用来表示逻辑运算（EQU、LT、LTE 和 NOTC）的结果，1 表示逻辑为真，0 表示逻辑为假。程序状态字中的前两个 bit 分别用来表示这两个标志。

程序计数器、指令寄存器和程序状态字作为三个专用寄存器，不能用来存放其他数据，否则不能保证目标代码正确执行。

另外，简单计算机的机器字长为 2 个字节。所以，通用寄存器、立即数和程序状态字用一个机器字（2 字节）来表示，程序计数器和指令寄存器用两个机器字（4 字节）来表示。

2) 存储器

系统的内存单元按字节进行编址。换句话说，内存中每个字节的存储单元都有一个区别于其他字节的地址编号。此简单计算机的内存空间为 2^{24} 个字节，地址编号从 0 到 $2^{24}-1$ 。

3) 端口

处理器在进行输入输出时通过端口来访问外围设备。Port #0 表示终端输入设备控制台，Port #15 表示终端输出设备控制台。端口号用一个机器字长（2 字节）来表示。

2 指令集

简单计算机的指令集见表 2，共 32 条指令和两条伪指令。32 条指令按功能划分为七种：控制指令(6 条)、堆栈操作指令(2 条)、数据存取指令(6 条)、I/O 操作指令(2 条)、算术运算指令(6 条)、位运算指令(6 条)和逻辑运算指令(4 条)。32 条指令按结构分为 8 类，因此书写格式有 8 类。指令长度固定为 32 位（bit），即 4 字节（Byte）。每条指令的前 5 位是指令对应的操作码，每个操作码用一个助记符来表示。伪指令用来定义数据，不生成机器码。

表 2 简单计算机指令集

类别	助记符	操作码	指令格式										格式类型
控制	HLT	00000	00000	000	0000	0000	0000	0000	0000	0000	0000	0000	1
	JMP	00001	00001	000	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	2
	CJMP	00010	00010	000	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	2
	OJMP	00011	00011	000	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	2
	CALL	00100	00100	000	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	2
	RET	00101	00101	000	0000	0000	0000	0000	0000	0000	0000	0000	1
堆栈	PUSH	00110	00110	RRR	0000	0000	0000	0000	0000	0000	0000	0000	3
	POP	00111	00111	RRR	0000	0000	0000	0000	0000	0000	0000	0000	3
数据存取	LOADB	01000	01000	RRR	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	4
	LOADW	01001	01001	RRR	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	4
	STOREB	01010	01010	RRR	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	4
	STOREW	01011	01011	RRR	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	AAAA	4
	LOADI	01100	01100	RRR	0000	0000	IIII	IIII	IIII	IIII	IIII	IIII	5
	NOP	01101	01101	000	0000	0000	0000	0000	0000	0000	0000	0000	1
I/O	IN	01110	01110	RRR	0000	0000	0000	0000	0000	PPPP	PPPP	PPPP	6
	OUT	01111	01111	RRR	0000	0000	0000	0000	0000	PPPP	PPPP	PPPP	6
算术运算	ADD	10000	10000	RRR	ORRR	ORRR	0000	0000	0000	0000	0000	0000	7
	ADDI	10001	10001	RRR	0000	0000	IIII	IIII	IIII	IIII	IIII	IIII	5
	SUB	10010	10010	RRR	ORRR	ORRR	0000	0000	0000	0000	0000	0000	7
	SUBI	10011	10011	RRR	0000	0000	IIII	IIII	IIII	IIII	IIII	IIII	5
	MUL	10100	10100	RRR	ORRR	ORRR	0000	0000	0000	0000	0000	0000	7
	DIV	10101	10101	RRR	ORRR	ORRR	0000	0000	0000	0000	0000	0000	7
位运算	AND	10110	10110	RRR	ORRR	ORRR	0000	0000	0000	0000	0000	0000	7
	OR	10111	10111	RRR	ORRR	ORRR	0000	0000	0000	0000	0000	0000	7
	NOR	11000	11000	RRR	ORRR	ORRR	0000	0000	0000	0000	0000	0000	7
	NOTB	11001	11001	RRR	ORRR	0000	0000	0000	0000	0000	0000	0000	8
	SAL	11010	11010	RRR	ORRR	ORRR	0000	0000	0000	0000	0000	0000	7
	SAR	11011	11011	RRR	ORRR	ORRR	0000	0000	0000	0000	0000	0000	7
逻辑运算	EQU	11100	11100	RRR	ORRR	0000	0000	0000	0000	0000	0000	0000	8
	LT	11101	11101	RRR	ORRR	0000	0000	0000	0000	0000	0000	0000	8
	LTE	11110	11110	RRR	ORRR	0000	0000	0000	0000	0000	0000	0000	8
	NOTC	11111	11111	000	0000	0000	0000	0000	0000	0000	0000	0000	1
伪指令	BYTE	无											
	WORD	无											

3 指令格式

指令操作码为 5bit, 用一个助记符来表示, 其余 27bit 分别用来表示寄存器的编号 (reg0、reg1、reg2)、立即数 (immediate)、数据在内存中的地址 (address)、外围设备的端口号 (port) 等, 为了补齐 27bit 还用到了填充位 (padding)。指令后 27bit 的解析按指令结构类型分为以下八种。

(1) op(5b) + padding(27)

此类指令有 4 个: HLT、RET、NOP、NOTC。以 NOTC 为例, 此类指令的书写形式为:

NOTC

此指令的机器码为:

1 1 1 1 1	0 0
op(5b)	pad(27b)

(2) op(5b) + padding(3b) + address(24b)

此类指令有 4 个: JMP、CJMP、OJMP、CALL。此类指令的书写时, 后面的地址用一个标号 label 来表示, 该标号必须在源码中定义, 进行汇编时转换为地址码。以 JMP 为例, 书写形式为:

JMP loop

此指令的机器码为:

0 0 0 0 1	0 0 0	A A
op(5b)	pad(3b)	address(24b)

(3) op(5b) + reg0(3b) + padding(24b)

此类指令有 2 个: PUSH、POP。以 PUSH 为例, 此类指令的书写形式为:

PUSH G

此指令的机器码为:

0 0 1 1 0	1 1 1	0 0
op(5b)	reg(3b)	pad(24b)

(4) op(5b) + reg0(3b) + address(24b)

此类指令有 4 个: LOADB、LOADW、STOREB、STOREW。此类指令在书写时, 后面的地址用一个符号来表示, 汇编时转换为地址码。以 LOADB 为例, 此类指令的书写形式为:

LOADB B cnt

此指令的机器码为:

0 1 0 0 0	0 1 0	A A
op(5b)	reg(3b)	address(24b)

(5) op(5b) + reg0(3b) + padding(8b) + immediate(16b)

此类指令有 3 个: LOADI、ADDI、SUBI。以 LOADI 为例, 此类指令的书写形式为:

LOADI A 8

此指令的机器码为:

0 1 1 0 0	0 0 1	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
op(5b)	reg(3b)	pad(8b)	immediate(16b)

(6) op(5b) + reg0(3b) + padding(16b) + port(8b)

此类指令有 2 个: IN、OUT。以 IN 为例, 此类指令的书写形式为:

IN D 0

此指令的机器码为：

0 1 1 1 0	1 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
op (5b)	reg (3b)	pad (16b)		port (8b)

(7) op(5b) + reg0(3b) + reg1(4b) + reg2(4b) + padding(16b)

此类指令有 9 个：ADD、SUB、MUL、DIV、AND、OR、NOR、SAL、SAR。以 ADD 为例，此类指令的书写形式为：

ADD A B C

此指令的机器码为：

1 0 0 0 0	0 0 1	0 0 1 0	0 0 1 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
op (5b)	reg (3b)	reg1 (4b)	reg2 (4b)	pad (16b)

(8) op(5b) + reg0(3b) + reg1(4b) + padding(20b)

此类指令有 4 个：NOTB、EQU、LT、LTE。以 NOTB 为例，此类指令的书写形式为：

NOTB B B

此指令的机器码为：

1 0 0 0 0	0 1 0	0 0 1 0	0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
op (5b)	reg (3b)	reg1 (4b)	pad (20b)	

此外，伪指令 BYTE 和 WORD 因为不生成机器码，所以没有对应的机器码格式。以 BYTE 为例，这两条指令的书写格式为：

```

BYTE    num
BYTE    num    =    0
BYTE    num[10]
BYTE    num[10] =    {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
BYTE    num[10] =    "This is a string"

```

4 指令功能

(1) 停机指令：HLT

功能：终止程序运行。

(2) 无条件转移指令：JMP label

功能：将控制转移至标号 label 处，执行标号 label 后的指令。

(3) 比较运算转移指令：CJMP label

功能：如果程序状态字中比较标志位 c 的值为 1(即关系运算的结果为真)，则将控制转移至标号 label 处，执行标号 label 后的指令；否则，顺序往下执行。

(4) 溢出转移指令：OJMP

功能：如果程序状态字中比较标志位 o 的值为 1(即算术运算的结果发生溢出)，则将控制转移至标号 label 处，执行标号 label 后的指令；否则，顺序往下执行。

(5) 调用子程序指令：CALL label

功能：将通用寄存器 A~G、程序状态字 PSW、程序计数器 PC 中的值保存到 ES，然后调用以标号 label 开始的子程序，将控制转移至标号 label 处，执行标号 label 后的指令。

(6) 子程序返回指令：RET

功能：将 ES 中保存的通用寄存器 A~Z、程序状态字 PSW 和程序计数器 PC 的值恢复，

控制转移到子程序被调用的地方，执行调用指令的下一条指令。

(7) 入栈指令: `PUSH reg0`

功能：将通用寄存器 `reg0` 的值压入堆栈 `SS`，`reg0` 可以是 `A~G` 和 `Z` 八个通用寄存器之一。

(8) 出栈指令: `POP reg0`

功能：从堆栈 `SS` 中将数据出栈到寄存器 `reg0`，`reg0` 可以是 `A~G` 七个通用寄存器之一，但不能是通用寄存器 `Z`。

(9) 取字节数据指令: `LOADB reg0 symbol`

功能：从字节数据或字节数据块 `symbol` 中取一个字节的数据存入寄存器 `reg0`，所取的字节数据在数据块 `symbol` 中的位置由寄存器 `G` 的值决定。用 `C` 的语法可将此指令的功能描述为：

`reg0 = symbol[G]`

例如，假设用伪指令定义了以下字节数据块 `num`：

`BYTE num[10] = {5,3,2,8,6,9,1,7,4,0}`

如果要将字节数据块 `num` 中第 5 个单元的值(即下标为 4 的元素)取到寄存器 `C`，指令如下：

```
LOADI    G    5
LOADB    C    num
```

后面的指令 `LOADW`、`STOREB` 和 `STOREW` 在操作上与此指令类似。

(10) 取双字节数据指令: `LOADW reg0 symbol`

功能：从双字节数据或双字节数据块 `symbol` 中取一个双字节的数据存入寄存器 `reg0`，所取的双字节数据在数据块 `symbol` 中的位置由寄存器 `G` 的值决定。

(11) 存字节数据指令: `STOREB reg0 symbol`

功能：将寄存器 `reg0` 的值存入字节数据或字节数据块 `symbol` 中的某个单元，存入单元的位置由寄存器 `G` 的值决定。用 `C` 的语法可将此指令的功能描述为：

`symbol[G] = reg0`

(12) 存双字节数据指令: `STOREW reg0 symbol`

功能：将寄存器 `reg0` 的值存入双字节数据或双字节数据块 `symbol` 中的某个单元，存入单元的位置由寄存器 `G` 的值决定。

(13) 取立即数指令: `LOADI reg0 immediate`

功能：将指令中的立即数 `immediate` 存入寄存器 `reg0`。立即数被当作 16 位有符号数，超出 16 位的高位部分被截掉。例如：

```
LOADI    B    65535
```

寄存器 `B` 的值为 -1。

```
LOADI    B    65537
```

寄存器 `B` 的值为 1。

(14) 空操作指令: `NOP`

功能：不执行任何操作，但耗用一个指令执行周期。

(15) 控制台输入指令: `IN reg0 0`

功能：从输入端口(即键盘输入缓冲区)取一个字符数据，存入寄存器 `reg0`。

(16) 控制台输出指令: `OUT reg0 15`

功能：将寄存器 `reg0` 的低字节作为字符数据输出到输出端口(即显示器)。

(17) 加运算指令: `ADD reg0 reg1 reg2`

功能：将寄存器 `reg1` 的值加上 `reg2` 的值，结果存入寄存器 `reg0`。如果结果超过 16 位

有符号数的表示范围，将发生溢出，使程序状态字的溢出标志位 o 置为 1；如果未发生溢出，则使程序状态字的溢出标志位 o 置为 0。

(18) 加立即数指令：ADDI reg0 immediate

功能：将寄存器 reg0 的值加上立即数 immediate，结果仍存入寄存器 reg0。如果结果超过 16 位有符号数的表示范围，将发生溢出，使程序状态字的溢出标志位 o 置为 1；如果未发生溢出，则使程序状态字的溢出标志位 o 置为 0。

(19) 减运算指令：SUB reg0 reg1 reg2

功能：将寄存器 reg1 的值减去 reg2 的值，结果存入寄存器 reg0。如果结果超过 16 位有符号数的表示范围，将发生溢出，使程序状态字的溢出标志位 o 置为 1；如果未发生溢出，则使程序状态字的溢出标志位 o 置为 0。

(20) 减立即数指令：SUBI reg0 immediate

功能：将寄存器 reg0 的值减去立即数 immediate，结果仍存入寄存器 reg0。如果结果超过 16 位有符号数的表示范围，将发生溢出，使程序状态字的溢出标志位 o 置为 1；如果未发生溢出，则使程序状态字的溢出标志位 o 置为 0。

(21) 乘运算指令：MUL reg0 reg1 reg2

功能：将寄存器 reg1 的值乘以 reg2 的值，结果存入寄存器 reg0。如果结果超过 16 位有符号数的表示范围，将发生溢出，使程序状态字的溢出标志位 o 置为 1；如果未发生溢出，则使程序状态字的溢出标志位 o 置为 0。

(22) 除运算指令：DIV reg0 reg1 reg2

功能：将寄存器 reg1 的值除以 reg2 的值，结果存入寄存器 reg0，这里进行的是整数除运算。如果寄存器 reg2 的值为零，将发生除零错。

(23) 按位与运算指令：AND reg0 reg1 reg2

功能：将寄存器 reg1 的值与 reg2 的值进行按位与运算，结果存入寄存器 reg0。

(24) 按位或运算指令：OR reg0 reg1 reg2

功能：将寄存器 reg1 的值与 reg2 的值进行按位或运算，结果存入寄存器 reg0。

(25) 按位异或运算指令：NOR reg0 reg1 reg2

功能：将寄存器 reg1 的值与 reg2 的值进行按位异或(按位加)运算，结果存入寄存器 reg0。

(26) 按位取反运算指令：NOTB reg0 reg1

功能：将寄存器 reg1 的值按位取反后，结果存入寄存器 reg0。

(27) 算术左移运算指令：SAL reg0 reg1 reg2

功能：将寄存器 reg1 的值算术左移 reg2 位，结果存入寄存器 reg0。在进行算术左移时，低位空位用 0 填充。

(28) 算术右移运算指令：SAR reg0 reg1 reg2

功能：将寄存器 reg1 的值算术右移 reg2 位，结果存入寄存器 reg0。在进行算术右移时，高位空位用符号位填充。

(29) 相等关系运算指令：EQU reg0 reg1

功能：将两个寄存器 reg0 和 reg1 的值进行相等比较关系运算：reg0 == reg1，关系运算的结果为逻辑真或逻辑假，存入程序状态字中的比较标志位 c。

(30) 小于关系运算指令：LT reg0 reg1

功能：将两个寄存器 reg0 和 reg1 的值进行小于关系运算：reg0 < reg1，关系运算的结果为逻辑真或逻辑假，存入程序状态字中的比较标志位 c。

(31) 小于等于关系运算指令：LTE reg0 reg1

功能：将两个寄存器 reg0 和 reg1 的值进行小于等于关系运算：reg0 <= reg1，关系运算的结果为逻辑真或逻辑假，存入程序状态字中的比较标志位 c。

(32) 比较标志位取反指令: NOTC

功能: 将程序状态字中的比较标志位 *c* 求反, 即将逻辑真变为逻辑假, 将逻辑假变为逻辑真。

(33) 字节数据定义伪指令: BYTE symbol[n] = {...} 蓝色字体部分为可选项

或: BYTE symbol[n] = "... " 蓝色字体部分为可选项

功能: 定义长度为 1 字节的字节型数据或数据块, 字节型数据块类似于 C 的字符数组。

(34) 字数据定义伪指令: WORD symbol[n] = {...} 蓝色字体部分为可选项

功能: 定义长度为 2 字节的双字节型数据或数据块, 双字节型数据块类似于 C 的整型数组(16 位系统)。

此外, 指令前可以加标号, 标号构成规则是标志符后加西文半角冒号 “:”, 标号必须具有唯一性, 不能重复定义。