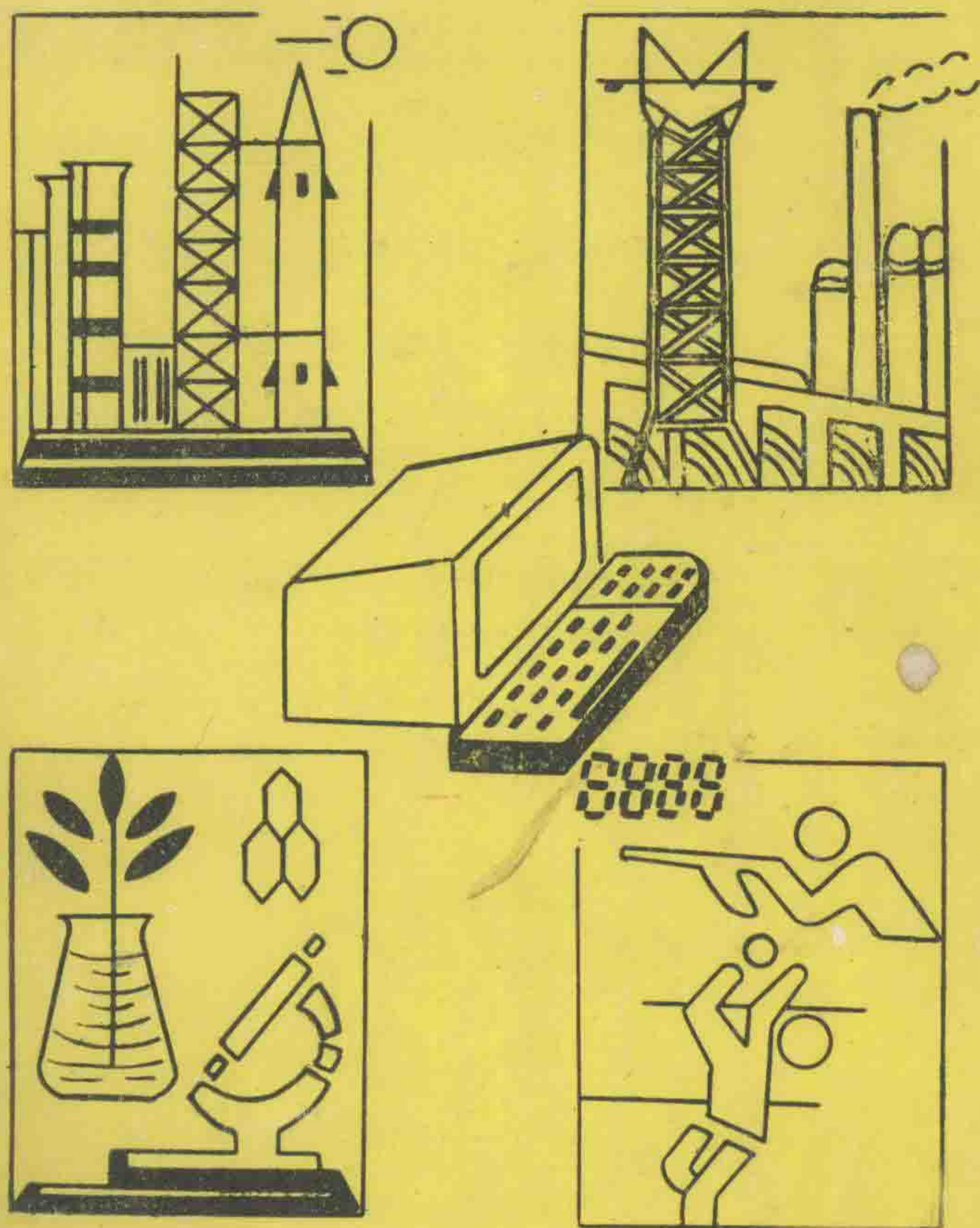


微型机开发与应用



1984年增刊

复旦大学计算机科学研究所
微机开发应用研究室

内 容 提 要

这书内容取材于实践,比较详细地介绍了MCS-48单片机汇编语言程序设计方法和技巧,每个程序都附有详细流程图,还列举了实用的键盘、显示、A/D、D/A等详细而正确硬件接线图。书中所列举的程序设计方法具有普遍意义。

本书是对从事微机应用、特别对智能仪器仪表、工业自动化控制中应用的工程技术人员是一本实用的参考书,也可作为微机初学者自学入门指导书。

本书还附上FD-35在线仿真器的监控仿真程序,对研制和开发新的微型机的工程技术人员有一定的参考价值。

本书即将由出版社正式出版,不得翻印

前 言

近几年来,微型电脑的应用发展很快,Intel公司的MCS—48单片机系列,在一块芯片上就集成了CPU, RAM, ROM, 定时/计数器和多种功能I/O, 具有体积小, 价格低, 控制功能强, 稳定可靠等优点, 广泛地用于生产过程的自动检测、实时控制和智能仪器仪表中, 是目前微机应用产品化最合适和应用最广泛的一种机种, 受到广大用户的欢迎。

由于单片机内存容量小, 不可能用高级语言来编制各种应用程序, 同时在有些实时控制中高级语言难于完成快速的实时控制, 这就要求用户使用汇编语言来编制程序。这种语言编制程序难度较大, 给单片机的推广普及带来困难, 我们在多次举办单片机应用短训班的基础上, 根据广大用户要求, 利用短短几个月的时间, 编制了一些应用性的实用子程序, 其中大部分程序是实际应用中的例子。本书为初学者介绍了一些汇编语言程序设计的基础和技巧, 每个程序都画出详细的流程图, 所提供的设计方法基本适用于8080/8085, Z80, 6800等各种微型机的汇编语言程序设计。

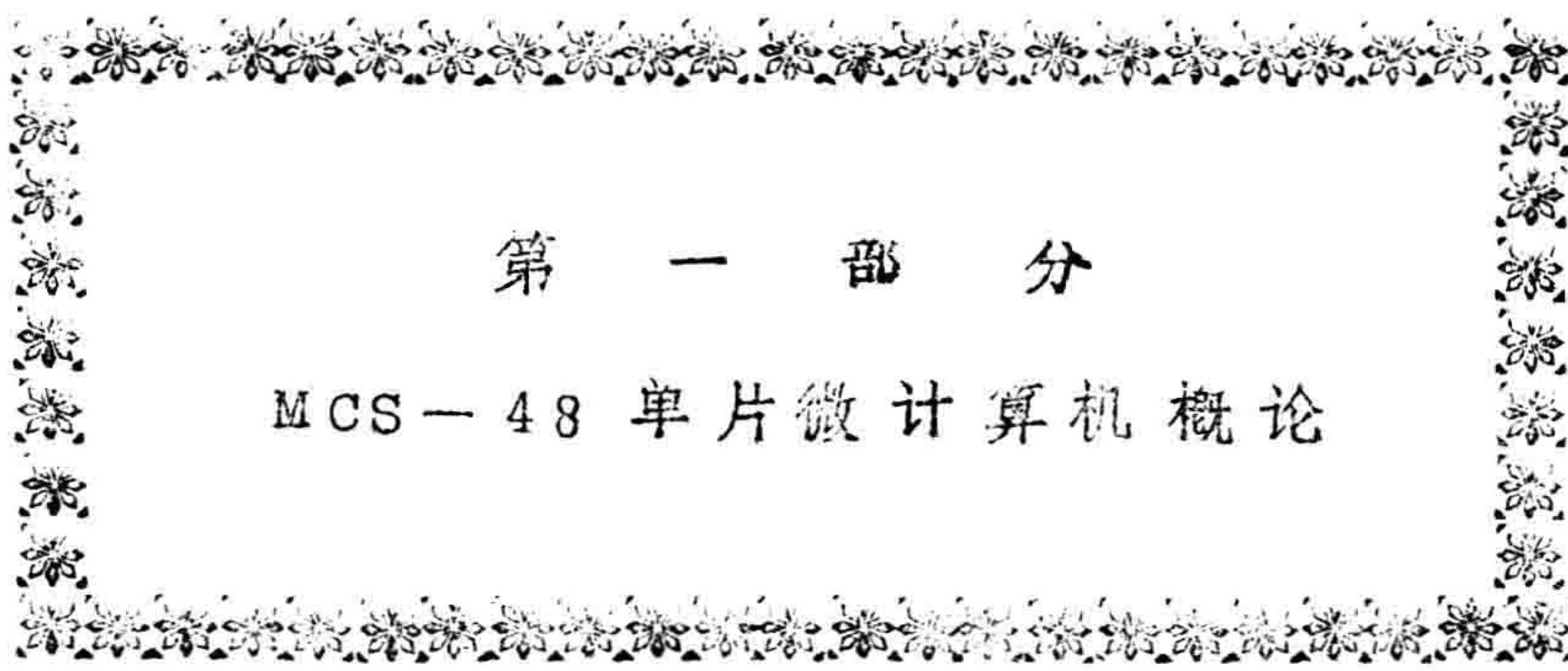
本书由姚志石、涂时亮同志编写。参加编写的还有张友德同志。高哲民也参加了部分编写工作。

在编写过程中, 受到徐君毅、陈章龙同志的热情帮助和指导, 薛剑虹同志为本书出版做了大量工作, 在此表示感谢。

由于编者水平有限, 时间仓促, 书中内容定有不当之处, 敬请读者批评指正。

编 者

1984年10月

A decorative rectangular border composed of a repeating floral pattern, enclosing the title text.

第 一 部 分

MCS-48 单片微计算机概论

目 录

第一部分 MCS-48 单片微计算机概论

第一章	MCS-48 简介	1
第二章	MCS-48 指令系统	6

第二部分 MCS-48 实用子程序

第一章	代码转换	10
1-1	十六进制→ASCII 码转换程序 HEASC ...	10
1-2	ASCII 码→十六进制数转换程序 ASCHE..	11
1-3	ASCII 码→BCD 码转换程序 ABCBC	12
1-4	BCD 码→ASCII 码转换程序 BCDAS	13
1-5	BCD 码→二进制转换程序 BCDB	14
1-6	十六位二进制数→BCD 数码的转换程序 1, BINBCD	16
1-7	十六位二进制数→BCD 数码的转换程序 2, BBCD	18
第二章	二进制定点运算	23
2-1	多字节加法程序 AADD 及减法程序 SUBA ...	23
2-2	单字节无符号数乘法程序 SEU	28
2-3	单字节有符号数乘法程序 SYU1	29
2-4	单字节有符号数乘法程序 SYU2	32
2-5	多字节无符号数乘法 MUX	35
2-6	多字节有符号数乘法 SFMUX	39
2-7	单字节无符号数除法 SDV	44
2-8	单字节有符号数除法 SYDIV	47
2-9	多字节无符号数除法 UXDV	50
2-10	多字节有符号数除法 UXDF	57
2-11	单字节的平方根 SQRS	61
2-12	三字节平方根的整数部分 SQRIT	63
第三章	BCD 运算	67

3-1	BCD码定点运算(寄存器间运算)	67
3-2	BCD码双字节加减法	68
3-3	BCD码多字节加减法内存运算 AAADD/ASUB	70
3-4	一位BCD码的乘法 ANDI	74
3-5	单字节BCD码乘法 ABCD	77
3-6	双字节BCD码乘法 BCDMUL	79
3-7	BCD码单字节除法 DIV	82
3-8	双字节BCD码除法 BUBBCD	85
3-9	单字节BCD码整数平方根	90
3-10	双字节BCD码整数平方根	92
第四章	A/D, D/A部分	94
4-1	A/D, D/A使用 I	94
4-2	A/D使用 II	104
第五章	浮点运算	110
5-1	对阶子程序 ADU	111
5-2	右规子程序 1, RST1	113
5-3	右规子程序 2, RST2	114
5-4	左规子程序 LST	115
5-5	浮点加法子程序 FADD	116
5-6	浮点乘法子程序 FMUL	119
5-7	浮点除法子程序 FDIV	120
5-8	正弦函数 Sinx 子程序 Sinx	123
5-9	多项式 $y = a_0 x^n + a_1 x^{n-1} + \dots + a_n$ 计算方法	127
5-9-1	主程序	129
5-9-2	取常数子程序 BA	133
5-9-3	数符阶符处理子程序 AH	134
5-9-4	乘法子程序(定浮点混合) ANBCD	135

5-9-5	加法子程序 BAND	138
5-9-6	四舍五入子程序 AAA	140
5-9-7	字节移位子程序 RRA	142
5-9-8	对阶处理程序 RABC	145
5-9-9	浮点加、减程序 RMA	149
5-9-10	取数子程序 AHR	152
5-9-11	减法子程序 SUBCD	154
5-9-12	移位子程序 RLA	156
5-9-13	传送子程序 RHA	158
5-9-14	左规子程序 RLR, RAB	159
第六章	键盘和显示及打印机的连接	164
6-1	键盘和显示部分程序 1	164
6-2	键盘和显示部分程序 2	175
6-3	显示子程序	187
6-4	GP-16 打印机接口	193
第七章	其他程序	206
7-1	实时钟程序 TCT	206
7-2	双字节定点数比较子程序 DCMF	209
7-3	多字节移位子程序 RRAA	210
7-4	多寄存内存移位子程序 RLR	212
7-5	找最大数子程序 SCH	214
7-6	顺序检索子程序 STE	216
第三部分 FD-35-II 型开发仿真程序		
附:	MCS-48 单片微计算机指令手册	218

1.2 程序存储器

内部程序存储器为 1024(8048) 或 2048(8049) 字节
8035/8039 内部无程序存储器, 必须使用外部器件。程序存储器中, 有三个单元特别重要。

1. 0号单元

处理机复位 (RESET) 时, 计算机总是从 0 号单元开始执行指令。

2. 3号单元

在允许中断时, 当中断输入线上有中断信号时计算机总是转到 3 号单元执行中断服务程序。

3. 7号单元

当允许定时/计数器中断时, 若定时器计数器溢出时, 则产生定时/计数中断, 计算机转到 7 号单元执行定时/计数器中断服务程序。

在使用时, 0 号单元应该放一条转移指令, 转到用户主程序; 3 号单元放转移指令, 使用中断时, 转到外部中断服务子程序上; 7 号单元的转移指令转到定时/计数器中断服务子程序上。

1.3 数据存储器

内部数据存储器有 64 / 128 字节

64 × 8 RAM 有三个地址范围 (0 ~ 7, 8 ~ 23, 24 ~ 31), 有专门的使用方法。

当初始通电时, 状态寄存器中的寄存器开关 BS 为 0。BS = 0, 则 0 ~ 7 就作为直接可寻址的高速暂存寄存器 R₀ ~ R₇, 这样指令 MOV A, R₇ 即把 RAM 中的第 7 号单元内容送到累加器 A。

可以使用寄存器区开关指令使 BS = 1, 当 BS = 1 时, 则把另 8 个单元 24 ~ 31 定义为 R₀ ~ R₇, 此时 MOV A, R₇ 即把内存 31 号单元内容送到累加器 A。8 ~ 23 号单元既可作一般的存储器, 也可作堆栈使用, 在调用子程序时用作记迹。可容纳 8 个返回地址, 故子程序的嵌套最大可以 8 级。除了 0 ~ 7, 24 ~ 31 可以直接寻

址外其它的内存单元的寻址只能通过R₀或R₁间接寻址。

1.4 输入/输出

有27根可用作输入或输出的线，它们是三个8位口，P₁口、P₂口、BUS口，P₁、P₂具有相同的特性，写到这些口的数据被锁存起来，输入时无锁存。

P₁、P₂为准双向口，允许每一根线作为输入，输出或同时作输入、输出，在这种用法时，由于它的内部特殊结构，在输入时必须先写一个“1”，否则输入要出错，当不同时作输入/输出时，无此要求。

P₁、P₂、BUS口还可以和立即数进行与/或操作。BUS口是真正双向口，输入/输出不能混用。

另有三个输入/输出线是：T₀、T₁、 $\overline{\text{INT}}$ 。能用条件转移指令来测试。

此外，T₀可作时钟输出

T₁作为计数输入

INT作中断输入

1.5 程序状态字

程序状态字(PSW)是一个8位状态字，发生子程序调用或中断时，PSW高四位和程序计数器PC一起被存入堆栈中(内存8~23号单元)，并能用RETR指令来恢复。

PSW各位定义如下：

位0~2，栈指针，初态为000，指向RAM8和RAM9号单元，第一个子程序调用或中断时把程序计数器的内容和PSW的高四位存入栈中。然后栈指针加1，指向10和11号单元。

子程序末尾应为返回指令，可使栈指针减1。

位3 不使用。

位4 工作寄存器区开关(BS)，0=区0，1=区1。

- 位 5 标志 F_0 ，它是用户控制的标志位
- 位 6 辅助进位 (AC)，由 ADD 指令产生，在十进调正指令 DAA 时使用。
- 位 7 进位位 (CY)，指示前面操作时累加器 A 溢出情况。

1.6 中断

可有二种中断，在 $\overline{\text{INT}}$ 上加一个低电平输入，可产生中断（在允许中断时），程序转到 003 号单元。当定时/计数器发生溢出时，也可产生中断，程序转到 007 号单元。

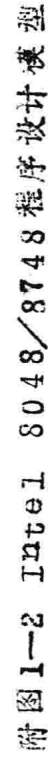
中断系统为单级中断，当检测到一个中断后，将不再理睬其他中断，只有执行 RETR 指令后，才重新允许中断输入，当中断系统检测到外部中断和定时/计数器中断同时存在时，则先响应外部中断。

1.7 复位与单步操作

复位时具有以下功能

1. 清 0 程序计数器，程序从 0 号单元开始执行。
2. 清 0 栈指针。
3. 清 0 寄存器区开关 $\text{BS} = 0$
4. 清 0 存储器区开关 $\text{DBF} = 0$
5. 关中断（定时和外部中断）
6. P_1 口， P_2 口为输入方式。
7. 清 0 F_0 ， F_1 标志位。
8. 不允许 CLK 从 T_0 输出。
9. BUS 口为高阻状态。
10. 停止计时。
11. 清 0 定时器标志。

单步功能给用户调试手段， $\overline{\text{SS}}$ 上低电平时，要求停止处理机运行，当 SS 升为高电平时，退出停止方式取下一个指令。



第二章 MCS-48 指令系统摘要

	助记忆符	说 明	字节	周期
累 加 器 类	* ADD A, Ri	寄存器加 A	1	1
	* ADD A, @Rj	数据存储器加 A	1	1
	* ADD A, #data	立即数加 A	2	2
	* ADDC A, Ri	寄存器带进位加 A	1	1
	* ADDC A, @Rj	数据存储器带进位加 A	1	1
	* ADDC A, #data	立即数带进位加 A	2	2
	ANL A, Ri	寄存器“与” A	1	1
	ANL A, @Rj	数据存储器“与” A	1	1
	ANL A, #data	立即数“与” A	2	2
	ORL A, Ri	寄存器“或” A	1	1
	ORL A, @Rj	数据存储器“或” A	1	1
	ORL A, #data	立即数“或” A	2	2
	XRL A, Ri	寄存器“异或” A	1	1
	XRL A, @Rj	数据存储器“异或” A	1	1
	XRL A, #data	立即数“异或” A	2	2
	INC A	A 加 1	1	1
	DEC A	A 减 1	1	1
	CLR A	A 清 0	1	1
	CPL A	A 取反	1	1
	DA A	A 十进制调整	1	1
	SWAP A	A 半字节交换	1	1
	RL A	A 左环移	1	1
	* RLC A	A 左环移带进位	1	1
	RR A	A 右环移	1	1
	* RRC A	A 右环移进位	1	1

以上指令中 R_i 为 $R_0, R_1, \dots, R_7$ 打 * 者指令影响 CY

R_j 为 R_0, R_1 , 运算结果均在 A

	助记忆符	说 明	字节	周期
输入	IN A, PK	口输入到 A	1	2
	OUTL PK, A	A 输出到口	1	2
	ANL PK, #data	口“与”立即数	2	2
	ORL PK, #data	口“或”立即数	2	2
	INS A, BUS	BUS 输入到 A	1	2
	OUTL BUS, A	A 输出到 BUS	1	2
	ANL BUS, #data	BUS“与”立即数	2	2
	ORL BUS, #data	BUS“或”立即数	2	2
	MOVD A, PH	扩展口输入到 A	1	2
	MOVD PH, A	A 输出到扩展口	1	2
寄存器类	ANLD PH, A	扩展口“与” A	1	2
	ORLD PH, A	扩展口“或” A	1	2
	以上指令中 PK 为 P ₁ 、P ₂ 口			
	PH 为扩展的 P ₄ 、P ₅ 、P ₆ 、P ₇ 口			
	INC Ri	寄存器加 1	1	1
	INC @Rj	数据存储器加 1	1	1
	DEC R	寄存器减 1	1	1
	JMP addr	无条件转跳	2	2
	JMPP @A	间接跳转	1	2
	DJNZ Ri, addr	寄存器减 1 非零跳	2	2
转移类	JC addr	当 Carry=1 时跳	2	2
	JNC addr	当 Carry=0 时跳	2	2
	JZ addr	当累加器为零时跳	2	2
	JNZ addr	当累加器不等于零时跳	2	2
	JTO addr	当 T0=1 时跳	2	2
	JNTO addr	当 T0=0 时跳	2	2
	JT1 addr	当 T1=1 时跳	2	2
	JNT1 addr	当 T1=0 时跳	2	2
	JF0 addr	当 F0=1 时跳	2	2

助 记 符	说 明	字 节	周 期
-------	-----	-----	-----

JF1	addr	当 F1=1 时跳	2	2
JTF	addr	定时器标志为 1 时跳	2	2
JNI	addr	当 $\overline{INT}=0$ 时跳	2	2
JBb	addr	累加器第 b 位为 1 时跳	2	2

以上转移指令除 JMP addr 外，均在指令的第 2 字节所在页内转，JMP 指令在 2K 内转，若从前 2K 转到后 2K 去，则先用存贮器区开关指令 SEL MBI，反之用 SEL MBO。

子程 序类	CALL	addr	转子程序	2	2
	RET		返回	1	2
	RETR		返回并恢复状态	1	2

CALL 指令的转移同 JMP 指令，但该指令不要出现在内存的 2046, 2047, 2048 单元处，同样也不要出现在 4093, 4094, 4095 处，因为返回时容易出错。

标 志 类	CLR	C	清除进位位	2	1
	CPL	C	进位位取反	1	1
	CLR	F0	清除标志 F0	1	1
	CPL	F0	标志 F0 相反	1	1
	CLR	F1	清除标志 F1	1	1
数 据 传 送 类	CPL	F1	标志 F1 取反	1	1
	MOV	A, Ri	寄存器送 A	1	1
	MOV	A, @Rj	数据存贮器送 A	1	1
	MOV	A, #data	立即数送 A	2	2
	MOV	Ri, A	A 送寄存器	1	1
	MOV	@Rj, A	A 送数据存贮器	1	1
	MOV	Rj, #data	立即数送寄存器	2	2
	MOV	@Rj, #data	立即数送数据存贮器	2	2
	MOV	A, PSW	PSW 送 A	1	1
	MOV	PSW, A	A 送 PSW	1	1

助 记 符	说 明	字 节	周 期
XCH A, R _i	A 和寄存器交换	1	1
XCH A, @R _j	A 和数据存储器交换	1	1
XCHD A, @R _j	A 和寄存器半字节交换	1	1
MOVX A, @R _j	外部数据存储器送A	1	2
MOVX @R _j , A	A 送外部数据存储器	1	2
MOVP A, @A	现行页面送A	1	2
MOVP3 A, @A	页面3送A	1	2

以上指令中 R_i 表示 R₀ ~ R₇, R_j 表示 R₀, R₁

定 时 器 / 计 数 器 类	MOV A, T	读定时器/计数器	1	1
	MOV T, A	写定时器/计数器	1	1
	STRT T	启动定时器	1	1
	STRT CNT	启动计数器	1	1
	STOP TCNT	停止定时器/计数器	1	1
	EN TCNT1	允许定时器/计数器中断	1	1
控 制 类	DIS TCNT1	禁止定时器/计数器中断	1	1
	EN I	允许外部中断	1	1
	DIS I	禁止外部中断	1	1
	SEL RB0	选寄存器堆0	1	1
	SEL RB1	选寄存器堆1	1	1
	SEL MB1	选存储区1	1	1
	ENTO CLK	允许TO作时钟输出	1	1
	NOP	空操作	1	1

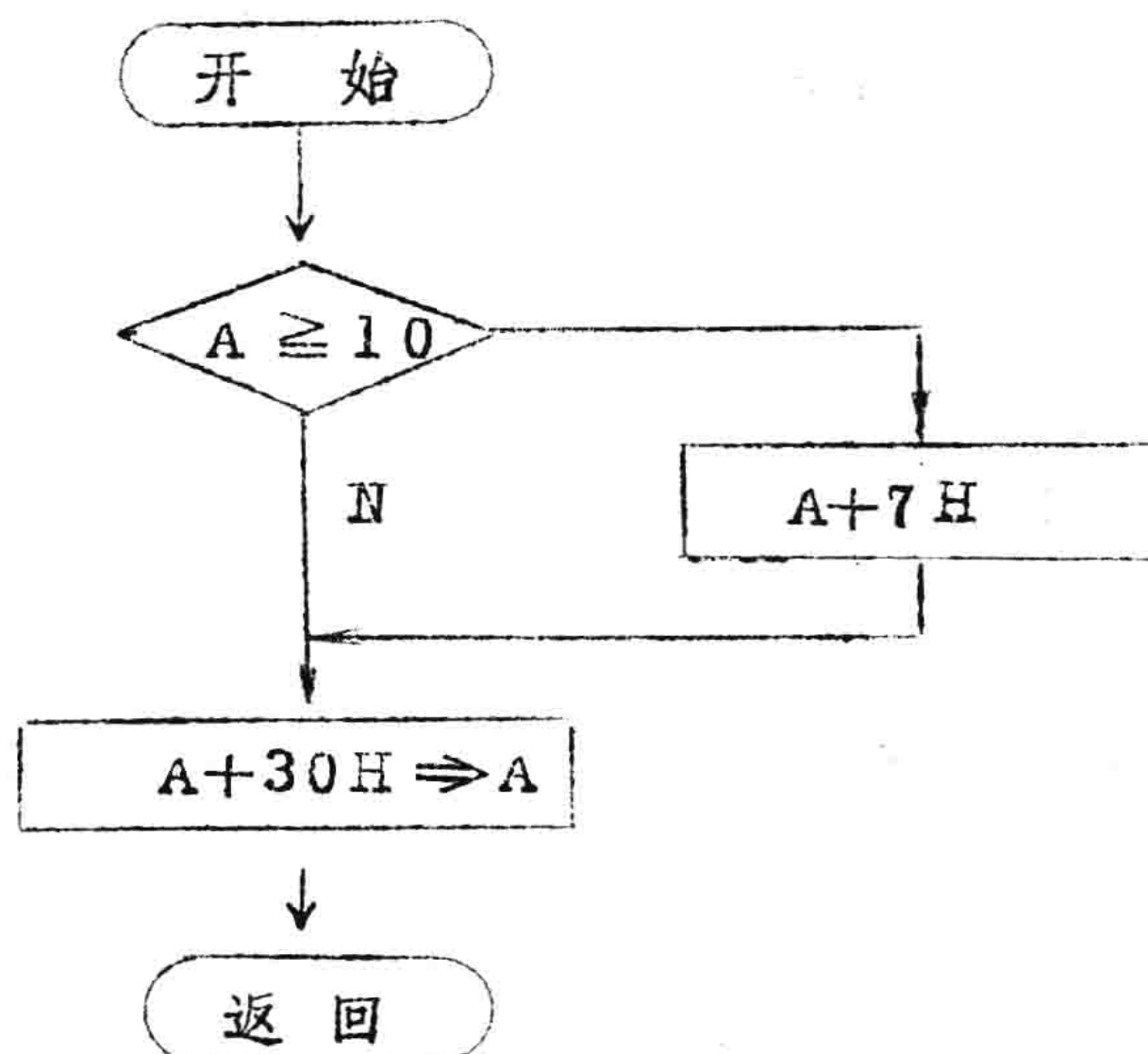
第一章 代码转换

1-1 十六进制 $\rightarrow$ ASCII码转换程序

入口参量：十六进制数在累加器中

出口参量：该数的ASCII码放在累加器A中

框图：



程序：

0/0	AA		HEASC:	MOV R2, A
1	03	F6		ADD A, #F6H
3	FA			MOV A, R2
4	E6	18		JNC AD30
6	03	07		ADD A, #07H
8	03	30	AD30:	ADD A, #30H
A	83			RET

说明：由于MCS-48中无比较指令也无减法指令，使用 $A-B = A + [B]$ 补，然后根据CY来判断：CY=1时， $A \geq B$ ；CY=0

时, $A < B$, 第二条指令中的 $F6H$ 即为 $[10]$ 补。

1-2 ASCII 码 $\rightarrow$ 十六进制

入口参量: ASCII 码在累加器 A 中

出口参量: 转移后的十六进制数在 A 中

入口: $ABCH$

方法: 若 A 中放的是 $0 \sim 9$ 的 ASCII 码, 则减去 $30H$ 后即得 $0 \sim 9$ 之间的数。若 A 中是 $A \sim F$ 的 ASCII 码, 则减去 $37H$ 后得 $A \sim F$ 之间的数。

框图:

