

# 微机原理补充

## 1 CPU模块

### 1.1 CPU寄存器

2812芯片大体上分为：中央处理单元（CPU），存储器和外设。其中的CPU寄存器如下（大致）：

#### CPU寄存器

- | - 读写改集成的累加器模块 ALU
  - | - 加法器  $ACC(32) = AH(16) + AL(16)$
- | - 32位乘法器模块
  - | - 乘数寄存器(暂存) : XT
  - | - 乘法寄存器(结果) :  $P(32) = PH(16) + PL(16)$
- | - 辅助寄存器模块 ARAU(32) : 地址寄存器算术单元
  - | - 辅助寄存器 AR(8个×32位) :  $XARn (n=0\sim7) = \text{高16位} + ARn (\text{低16位})$
  - | - 数据页指针 DP(16) : 直接寻址(22,4M) = DP(16) + 偏移量(6)
- | - 堆栈指针 SP(16) : 偏移量小于64, 只能访问[0,0xffffH]低地址空间单元, 复位0x0400
- | - 程序计数器 PC : 总是包含到达D2阶段指令的地址
- | - 指令计数器 IC : 装入下一条指令的地址, 保持到下个D2阶段
- | - 状态寄存器 ST0/ST1
  - | - 溢出位 V : 1为有溢出, 用于判断有符号运算是否出错
  - | - 进位/借位位 C : 用于判断无符号数高低, 对有符号数而言不重要
    - | - 0 : 无进位/有借位
    - | - 1 : 有进位/无借位
  - | - 负标志位 N : 1为有负数产生, 用于有符号数判断大小
  - | - 零标志位 Z : 1为有0产生
  - | - 符号扩展模式位 SXM : 0-无扩展, 1-符号扩展
  - | - (ST1) - 辅助寄存器指针 ARP
- | - 其他 (如中断控制寄存器 IFR, IER等)

### 1.2 总线

2812的存储器空间被分为了两大块：**程序空间（P）和数据空间（D）**。

访问任一空间，都需要两种总线配合：**地址总线（A）和数据总线（D）**，前者终于传送存储单元的地址，后者用于传送存储单元的具体内容。

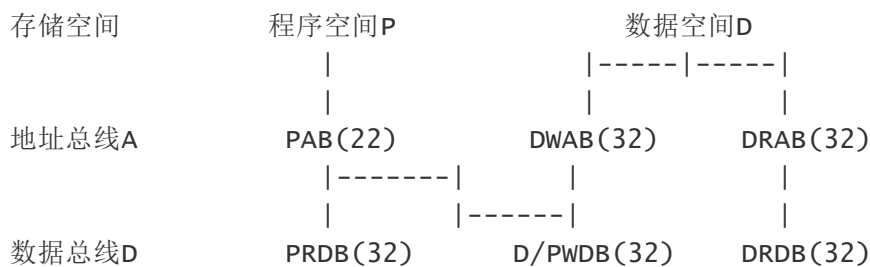
- 下面对英文缩写做一点简单的说明：

P : Program 程序(空间)  
A : Address 地址(总线)  
D : Data 数据(空间或总线)  
R/W : Read/Write 读取/写入 省略表示两者均可  
B : Base 总线

- 可以看到：

1. 一共有6跟总线，三根数据总线，3根地址总线
2. 地址总线A少一根的原因是程序空间P的读写共用一根总线（因此程序的读写不能同时进行）
3. 数据总线D少一根的原因是程序、数据空间共用一根总线进行数据的写入

- 下面就是2812CPU对存储空间内数据的调用的过程



## 1.3 指令流水线

2812的指令流水线大体上遵循以下几点规则：

1. 一条指令最多分8步完成
2. 每一步都要一个时间间隔完成
3. 同一时间最多可能有8条指令在执行
4. 步与步之间可能会插入间隔（如当此步需要用到上一步的结果，就会等待）

流水线的8步分别为：

获取指令地址 → 获取指令内容 → 对指令进行解码 → 解析操作数地址  
 → 锁定操作数地址 → 获取操作数 → CPU执行 'real work' → 将结果存入到内存

## 2 2812指令系统——寻址

### 2.1 寻址的基本操作

2812采用增强型哈佛总线结构，能够并行的访问地址和数据存储空间，其寻址的范围为

`[0, 0x3fffff]`（即 $2^{22}=4M$ ）。介绍寻址之前，大致先了解一下 `MOV` 指令，大意就是：`MOV A B` 近似于把B放到A里面，这也是最重要的汇编语言之一。

寻址方式大致上的分类有以下3种：

寻址方式	含义	写法
立即寻址	直接对应数字本身	#
直接寻址	某一个单元格内对应的数字	@
间接寻址	指向某个单元的指针对应的单元中的数字	*

下面介绍一些常用的寻址方法及其写法（下文中 `loc16` 表示任一16位的地址/数据存储单元）：

#### 1. 立即数寻址（把数据送到寄存器）

- (a) 数据不能太大，一般为16bit，只有XAR可以接受22bit的数据
- (b) ACC不能直接接受更多位的立即数

```

1 | MOVL XARn, #22bit
2 | MOVW DP, #16bit
3 | MOVW DP, #16bit >> 6
4 | MOV SP, #16bit
5 | MOV ACC, #16bit

```

#### 2. 偏移量直接寻址（DP的使用，好用的）

```

1 MOVW DP, #0x70D0 >> 6 // 把70D0“赋值”给DP
2 MOV @4, #16bit // 偏移量=4, 把某个数16bit“赋值”给70D4单元

```

因为DP寻址方式是用16位DP“并上”6位偏移量来实现的，因此为了避免繁杂的计算，在给DP输入时要把 `0x70D4` 右移6位，直接移到偏移量的位置上，就可以把“并”换成简单的加法。

### 3. 堆栈间接寻址（SP的使用）

```

1 MOV SP, #0x70DF // 把70DF“赋值”给SP
2 MOV *-SP[5], #16bit // 偏移量=-5, *表示指针, 把某个数16bit“赋值”给70DA单元

```

SP就是只能用负的偏移量，也就是 `-SP[n]`，别问为什么

### 4. 寄存器间接寻址（AR/XAR的使用）

```

1 MOV XAR6, #0x70D0 // 把70D0“赋值”给XAR6
2 MOV *+XAR6[4], #16bit // 偏移量=4, 把某个数16bit“赋值”给70D4单元

```

AR/XAR就是只能用正的偏移量，也就是 `+XARn[m]`，别问为什么

上述方法2/3/4的第二行都属于 `MOV loc16 #16bit` 的形式

### 5. 寄存器直接寻址（说实话我看不太懂）

```

1 MOVL XAR2, @ACC // MOV loc16, AX
2 MOVL ACC, @XAR2 // MOV AX, loc16

```

之前讲过，ACC(32)=AH(16)+AL(16)，这里用AX指代AH或AL中的任意一个。AX是汇编里面比较特殊的一个量，其他的建议别管这个寻址方式

### 6. 空间立即寻址

```

1 MOV *(0:16bit), loc16 // 这个是通式
2 MOV *(0:0x70D4), @AX // 这个是例子

```

- 以下是两种被禁止的寻址写法：

- `MOV loc16 loc16`，代表为：`MOV DP XAR`，但是 `ACC(AH/AL)` 除外
- `MOC *(0:0x16bit), #16bit`，代表为：`MOV *(0:0x70D4), 0xFF00`

## 2.2 寻址例程

```

1 // Gpfmux=0xF0FF (Gpfmux地址为0x70D4)
2 /* Code 1 */
3 MOVW DP, #0x70D0 >> 6
4 MOV @4, #0xF0FF
5 /* Code 2 */
6 MOV AL, #0xF0FF
7 MOV *(0:0x70D4), @AL // 利用中间变量AL进行空间立即寻址
8 /* Code 3 */
9 MOVL XAR2, #0x70D0
10 MOV *+XAR2[4], #0xF0FF

```

## 3 汇编语言

### 3.1 常用汇编语言

#### 1. 传送指令

```
1 MOV / MOVL (32bit) / MOVW (专用于DP) / MOVU (高位0扩展, 常用于ACC)
```

#### 2. 简单指令集 (AX=AH或AL)

```
1 /* 加法 */ ADD AX, loc16 // AX = AX + loc16
2 /* 减法 */ SUB AX, loc16 // AX = AX - loc16
3 /* 比较 */ CMP AX, loc16 // 根据比较结果置位, 但不会改动数值
4 // 如置零位ZF, 符号位SF
5 /* 与 */ AND AX, loc16 // AX = AX & loc16
6 /* 或 */ OR AX, loc16 // AX = AX | loc16
7 /* 取反 */ NEG AX, loc16 // AX = -AX
```

上面的指令后面多一个B表示短指令, 如 `ANDB AX #8bit`

#### 3. 移位指令 (可以代替以2为倍数的乘法)

```
1 LSL AX, #16bit // 逻辑左移(unsigned)
2 LSR AX, #16bit // 逻辑右移(unsigned)
3 ASR AX, #16bit // 算数右移(signed), 保留符号位
4 // 例: LSR AL, #4 表示 AL = AL / 16
```

#### 4. 重复执行 (常用于除法, 要背除法的代码段)

```
1 RPT #N-1 // 重复下一行N次
2 || .... // 只能写简单的代码, 比如加、减等
```

#### 5. 乘法 (写法很多, 只介绍其中一种)

```
1 MPYB P,T,#8bit // P (signed 32) = T (signed 16) * 8bit (unsigned 8)
2 // P (32) = PH (16) + PL (16)
```

#### 6. 条件减法 (常用于除法, 要背除法的代码段)

```
1 SUBCU ACC, loc16 // ACC = ACC 条件减 loc16, 具体原理不想写了
```

#### 7. 条件指令 (好多啊, 这咋记得住啊)

```
1 B CODE, LOGIC // B代表条件指令, SB短跳8bit, LB长跳22bit
2 // CODE是代码块的名称, 可以理解为C或Py的函数
3 // LOGIC表示条件名, 满足跳转, 不满足不跳
```

条件列表:

中文名	条件名	翻译	具体判断标准
不等于	NEQ	Not Equal To	Z = 0
等于	EQ	Equal To	Z = 1
大于	GT	Greater Then	Z = 0 AND N = 0
大于等于	GEQ	Greater Then Or Equal To	N = 0
小于	LT	Less Then	N = 1
小于等于	LEQ	Less Then Or Equal To	Z = 1 OR N = 1
高于	HI	Higher	C = 1 AND Z = 0

高于等于	HIS,C	Higher Or Same, Carry Set	C = 1
低于	LO,NC	Lower, Carry Clear	C = 0
低于等于	LOS	Lower Or Same	C = 0 OR Z = 1
未溢出	NOV	No Overflow	V = 0
溢出	OV	Overflow	V = 1
我不到啊	NTC	Test Bit Not Set	TC = 0
我不到啊	TC	Test Bit Set	TC = 1

8. 自增 (i++, 这个不用背)

```
1 | INC loc16 // 如 INC @2
```

## 3.2 汇编代码块

1. 除法 (需要记)

```
1 | /* 无符号数除法 */
2 | /*
3 |     Num ÷ Den = Quot ... Rem
4 |     商: Quot = Num / Den
5 |     余数: Rem = Num % Den
6 | */
7 | MOVU ACC, @Num // AH = 0, AL = Num,
8 | RPT #15 // 16bit, Repeat 15 times
9 | || SUBCU ACC, @Den // 条件减法
10 | MOV @Rem, AH // 余数存在高位AH, 移到Rem里
11 | MOV @Quot, AL // 商存在低位AL, 移到Quot里
```

2. 代码块的调用

```
1 | /* 代码块的调用—类似于C与Py的函数 */
2 | TODO // PRE CODE
3 | LC NAME // 调用, LC = CALL
4 | TODO // POST CODE
5 |
6 | NAME:
7 |     TODO // FUNCTION
```

3. 考试的代码块要求

```
1 | /* 考试的代码块要求 */
2 | /*
3 |     1. 有标号, 或者“函数名”
4 |     2. 以LRET结束
5 | */
6 | /* 示例 */
7 | TEST:
8 |     TODO // Code Here
9 | LRET
```

### 3.3 汇编实例

#### 例1

```
1  /*
2   将9000H单元中的16进制数(<99)转为8421BCD码存入9002H单元中
3   如: 51H = 81 -> 81H
4   实现方法: 51H/10 = 8...1 -> 8*16+1 = 81
5  */
6   MOVW DP, #0x9000 >> 6
7   MOVU ACC, @0
8   LC HEX2BCD
9   MOV @2, AL
10
11  HEX2BCD:
12   MOV T, #10
13   RPT #15
14   || SUBCU ACC, @T // 除法: 高位AH=余数, 低位AL=商
15   LSL AL, #4      // AL = AL(3..0) * 16
16   ADD AL, @AH     // AL = AL + AH
17   LRET
```

#### 例2

```
1  /*
2   将9004H单元中的8421BCD码转为16进制数存入9006H单元
3   如: 56H -> 56 = 38H
4   实现方法: 56H -> 5*10+6 = 56 = 38H
5  */
6   BCD2HEX:
7   MOVW DP, #0x9000 >> 6
8   MOVU ACC, @4 // ACC = 0 0 | 5 6
9   MOV AH, @AL // ACC = 5 6 | 5 6
10  ASR AH, #4 // AH = 0 5
11  AND AL, #0x0F // AL = 0 6
12  MOV T, @AH
13  MPYB P,T,#10 // P = AH * 10
14  MOV AH, @PL // AH = 50
15  ADD AH, @AL // AH = 50 + 6
16  MOV @6, AH
17  LRET
```

#### 例3 (这个太难了考试不会考的)

```
1  /*
2   数组求和, 求给定数组前10个元素之和, C代码如下:
3  */
4  int k = 0,i;
5  int m[10];
6  for (i=0;i<10;i++) k += m[i];
7  /* 下面是汇编语言写法 */
8  /* int k = 0,i; */
9  MOVW DP, #0x3F9000 >> 6 // 设i为@0, k为@1
10  MOV @1, #0 // k = 0
```

```

11  /* int m[10] */
12  MOV XAR4, #0x3F9040 // 设为数组初始地址
13  /* for(i=0;i<10;i++) k += m[i]; */
14  MOV @0, #0 // i = 0
15  MOV AL, @0
16  L1:
17  MOV ACC, @0
18  ADDL @XAR4, ACC // XAR4 = &m[i]
19  MOV AL, *+XAR4[0] // AL = m[i]
20  ADD @1, AL // k = k + m[i]
21
22  INC @0 // i++
23  MOV AL, @0 // AL = i
24  CMPB AL, #10
25  SB L1, LT // LT = Less Than
26  LRET

```