

# 汇编指令集

## 第一章寄存器

### **pdata:**

属性: RW

说明: 通用寄存器, 72bit。执行 **fetch** 与 **ifetch** 指令时会默认放入该寄存器中。

备注: **blank** 标志位时通过该寄存器中的值来

示例:

**setarg 5**           //将 5 放入 pdata 寄存器中

**fetch 1,0x400**   //将内存地址为 0x400 的数据取出放入 pdata 中。

**ifetch 1,contr**   //将 contr 中所存地址中的数据取出放入 pdata 中。

### **temp:**

属性: RW

说明: 通用寄存器, 64bit

备注: none

示例:

**arg 5,temp**           //将 5 放入 temp

**fetcht 1,0x00**       //从内存 0x00 中取一个字节放入 temp

**arg 0x1,contr**

**ifetcht 1,contr**       //从内存 0x01 中取一个字节放入 temp

**arg 0x2,contw**

**ifetcht 1,contw**   //从内存 0x02 中取一个字节放入 temp

.....

## **rega:**

属性: RW

说明: 通用寄存器, 32bits

备注: none

示例:

```
arg 0x01,rega          //  
copy rega,pdata        //将 rega 中的数值拷贝到 pdata 中  
ifetch 1,rega           //将内存 0x01 中一个字节取出放入 pdata
```

.....

程序分析: 在使用时要注意 `ifetch 1,rega` 是将 `rega` 中所存地址中的一个字节取出, 而不是将 `rega` 中的数值取出。

## **regb:**

属性: RW

说明: 通用寄存器, 32bits

备注: none

示例:

```
arg 0x01,regb          //  
copy regb,pdata        //将 regb 中的数值拷贝到 pdata 中  
ifetch 1,regb           //将内存 0x01 中一个字节取出放入 pdata
```

.....

程序分析: 在使用时要注意 `ifetch 1,regb` 是将 `regb` 中所存地址中的一个字节取出, 而不是将 `regb` 中的数值取出。

## **regab :**

属性: RW

说明: rega 与 regb 的合并的寄存器, 64bits, rega 为高 32 位, regb 为低 32 位。

备注: none

示例:

```
arg 0x01,regab      //  
copy regab,pdata    //将 regab 中的数值拷贝到 pdata 中  
ifetch 1,regab      //将内存 0x01 中一个字节取出放入 pdata
```

.....

程序分析: 在使用时要注意 ifetch regab 是将 regab 中所存地址中的一个字节取出, 而不是将 regab 中的数值取出。

## **regc:**

属性: RW

说明: 通用寄存器, 17bits

备注: none

示例:

```
arg 0x01,regc      //  
copy regc,pdata    //将 regc 中的数值拷贝到 pdata 中  
ifetch 1,regc      //将内存 0x01 中取出一个字节放入 pdata
```

.....

程序分析: 在使用时要注意 ifetch 1,regc 是将 regc 中所存地址中的一个字节取出, 而不是将 regc 中的数值取出。

## **queue:**

属性: RW

说明：搭配 `qset,qisolate` 等指令使用

备注：none

示例：

```
arg 0,queue           //将 0 放入 queue 中
arg 100,loopcnt
setarg 0
qset1 pdata           //将 pdata 寄存器中第 0 个 bit 置 1，所
置的 bit 位为 queue 的值
hstore 1,USB_TRG      //将 pdata 中的值存入硬件寄存器中，
所以使用 hstore
.....
```

## **contr:**

属性：RW

说明：读指针，指向下一个读内存地址，执行读内存操作后会根据读出长度自动移动指针。

备注：`fetch`，`ifetch` 等指令会影响该寄存器值。

示例：

```
arg mem_rxbuf,contr    //将变量 mem_rxbuf 的地址放入寄存器
contr 中
ifetch 1,contr         //从寄存器 contr 中所存地址的数据取出
.....
```

## **contw:**

属性: RW

说明：写指针，指向下一个写内存地址，执行写内存操作后会根据写入长度自动移动指针。

备注：store，istore 等指令会影响该寄存器值。

示例：

setarg 5

arg mem\_rxbuf,contw       //将变量 mem\_rxbuf 的地址放入寄存器  
contw 中

istore 1,contw               //将 pdata 寄存器中的值存入 contw 中所  
存的地址中

.....

## **contru:**

属性：RW

说明：读指针，指向下一个读内存地址，执行读内存操作后会根据读出长度自动移动指针。

该指针为 Uart 串口使用的，且该指针为可回环指针。

备注：fetch，ifetch 等指令会影响该寄存器值。

示例：

ifetch 1,contru           //从寄存器 contru 中所存地址的数据取出

.....

## **contwu:**

属性：RW

说明：写指针，指向下一个写内存地址，执行写内存操作后会根据写入长度自动移动指针。

该指针为 Uart 串口使用的，且该指针为可回环指针。

备注：store，istore 等指令会影响该寄存器值。

示例：

istore 1,contwu //将 pdata 寄存器中的值存入 contwu 中  
所存的地址中

.....

## loopcnt:

属性: RW

说明: 循环计数器, 配合 loop 使用, 每次 loop 跳转 loopcnt 自动-1, loopcnt 等于 0 时, loop 指令不再跳转而顺序执行。

备注: loopcnt 也可作为普通寄存器使用, 但要注意 loop 会影响到它。

示例:

arg 20,loopcnt

dsc\_string2\_short\_loop:

hjam 1,USB\_TRG

loop dsc\_string2\_short\_loop//hjam 1,USB\_TRG 会被执行 20 次

rtn

## mark:

属性: RW

说明: 通用寄存器, 64bits

备注: none

示例:

set0 3,mark //将 mark 寄存器的第 3 位置 0

rtnmark0 11 //判断 mark 寄存器的第 11 位是否为 0, 若为 0 则返回。

.....

## null:

属性: W

说明：通用寄存器，64bits。数据回收工作，在需要标志位时而不需要运算结果时使用。

备注：none

示例：sub pdata,1,null      //将 1-pdata 的结果放到 null 中  
      rtn zero

## **clkn\_bt:**

属性：R

说明：时间寄存器，64bits。会随着芯片 runing 不断自增。达到最大值则从零开始。

备注：clkn 时钟为 master 时钟，slave 以该时钟为校正时钟。

示例：

deposit clkn\_bt                      //将 clkn\_bt 内时间数值取出放入 pdata  
store 4,mem\_clkn\_bt  
.....

## **clkn\_rt:**

属性：R

说明：时间寄存器，会随着芯片 runing 不断自增。达到最大值则从零开始。

备注：与 clkn\_bt 精度不同，clkn\_rt 精度更高。

示例：

until clkn\_rt,meet //等待 clkn\_rt 将 meet 标志位置 1。  
.....

## **clke\_bt:**

属性：R

说明：时间寄存器，64bits。会随着芯片 runing 不断自增。达到最大值则从零开始。

备注：clke 时钟为 slave 时钟，在 ble 蓝牙中为链接开始时开始计时。

示例：

```
deposit clke_bt          //将 clke_bt 内时间数值取出放入 pdata
store 4,mem_clkn_bt
```

.....

## **clke\_rt:**

属性：R

说明：时间寄存器，会随着芯片 runing 不断自增。达到最大值则从零开始。

备注：与 clke\_bt 精度不同，clke\_rt 精度更高。

示例：

```
until clke_rt,meet //等待 clke_rt 将 meet 标志位置 1。
```

## **pc:**

属性：RW

说明：程序指针，指向下一条要执行的指令。

备注：常做跳转使用。

示例：

```
setarg le_mouse
copy pdata,pc          //rtn through cb functon.
```

.....

程序分析：该段程序主要是完成一个跳转功能，将函数 le\_mouse 的 pc 排号放入到 pc 中当程序执行完该代码后会直接跳转到 le\_mouse 函数中。



.....

## 第二章标志位

### **blank:**

属性：R

说明：pdata 无数据或值为零时置 1；否则清 0。

备注：在使用 fetch 后该标志位才有可能会被置 1。

示例：

```
fetch 1,mem_app_evt_timer_count
```

```
rtn blank
```

.....

程序分析：在 mem\_app\_evt\_timer\_count 变量中取一个字节放到 pdata 中，如果 pdata 中值为 0，则 blank 标志位会被置 1，会执行 rtn。

### **positive:**

属性：R

说明：符号志位，使用 ALU 时运算结果大于等于零时置 1；否则清 0。

备注：常用于 isub 与 sub。

示例：

```
sub pdata,17,null
```

```
rtn positive
```

```
fetch 4,mem_next_btclk
```

```
isub temp,pdata
```

nrtn positive

.....

程序分析：首先，第一个 sub 是做 17-pdata 操作如果结果大于等于 0 则 positive 会被置 1 会执行 rtn。lsub 是做 pdata-temp 操作如果结果大于等于 0 则 positive 会被置 1 会执行 rtn。

## true:

属性：R

说明：逻辑运算标志位。逻辑运算结果为真时置 1；否则清 0。

备注：常用于 compare 与 icompare。

示例：

fetch 1,mem\_device\_option

isolate1 1,pdata //如果 pdata 第一个 bit 为 1 则  
true 置 1 否则为 0

call app\_ble\_start\_adv,true

.....

fetch 1,mem\_state\_map

compare 0x0,pdata,0xc0 //比较 0x0 与 pdata 中第 6、7bit 的  
值，相等 true 被置 1

branch quit\_connection\_not\_clear\_mark,true

.....

fetcht 1,mem\_temp\_arq

fetch 1,mem\_arq

icompare 0x04,temp //比较 pdata 与 temp 的第 2 个 bit 相等 true 被  
置 1

nbranch rx\_type\_dispatch,true

.....

## **user:**

属性：RW

说明：用户标志位，开发者自行对其置 1 清 0 和判断，以实现程序的逻辑处理。

备注：none

示例：

```
enable user      //user 置 1
```

```
disable user     //user 置 0
```

.....

## **user2:**

属性：RW

说明：用户标志位，开发者自行对其置 1 清 0 和判断，以实现程序的逻辑处理。

备注：none

示例：

```
enable user2     //user 置 1
```

```
disable user2    //user 置 0
```

.....

## **user3:**

属性：RW

说明：用户标志位，开发者自行对其置 1 清 0 和判断，以实现程序的逻辑处理。

备注：none

示例：

```
enable user3    //user 置 1
disable user3   //user 置 0
.....
```

## **zero:**

属性：R

说明：零标志位，alu 运算结果为 0 时置 1；否则清 0。

备注：常用于 isub 与 sub。

示例：

```
ifetch 6,contr
    isub temp,null
    rtn zero
fetch 1,mem_switch_fail_master_count
    sub pdata,1,null
branch app_bt_role_switch,zero
.....
```

## **master:**

属性：RW

说明：用户标志位，开发者自行对其置 1 清 0 和判断，以实现程序的逻辑处理。

备注：none

示例：

```
enable master  //master 置 1
disable master //master 置 0
.....
```

## **slave:**

属性：RW

说明：用户标志位，开发者自行对其置 1 清 0 和判断，以实现程序的逻辑处理。

备注：none

示例：

```
enable slave    //slave 置 1
```

```
disable slave   //slave 置 0
```

.....

## **wake:**

属性：RW

说明：用户标志位，开发者自行对其置 1 清 0 和判断，以实现程序的逻辑处理。

备注：none

示例：

```
enable wake     //wake 置 1
```

```
disable wake    //wake 置 0
```

.....

## **match:**

属性：RW

说明：用户标志位，开发者自行对其置 1 清 0 和判断，以实现程序的逻辑处理。

备注：none

示例：

enable match    //match 置 1

disable match    //match 置 0

.....

## **timeout:**

属性： R

说明： 时间标志位， 由定时器的时间检测的结果， 时间间隔到时置 1， 否则置 0。

备注： none

示例：

setarg 1000

iforce stop\_watch

    until null,timeout

    rtn

.....

程序分析： 将 1000 放入定时寄存器 stop\_watch 中， pc 会停在 until null,timeout 直到 timeout 被置 1。

## **sync:**

属性： R

说明： 硬件标志位， 接收数据包时， 可以找到时置 1； 否则清 0。

备注： none

示例：

correlate null,timeout

copy clke,temp

    storet 6,mem\_sync\_clke

nbranch end\_of\_packet, sync //sync 置 1 则跳转到  
end\_of\_packet

.....

## le:

属性: RW

说明: 用户标志位, 开发者自行对其置 1 清 0 和判断, 以实现**ble**设备的区分。

备注: none

示例:

enable le //le 置 1

disable le //le 置 0

## 第三章逻辑运算指令

### and:

格式: and , ,

参数: immediate 为立即数(最大为 9bit)

regr 为源寄存器

regw 为目标寄存器

描述:  $\text{regw} = \text{regr} \& \text{immediate};$

示例: `and clkn_bt,0x1fc,bt_clk //bt_clk = clkn_bt&0x1fc`

备注:none

## **and\_into:**

格式: `and_into ,`

参数: immediate 为立即数(最大为 9bit)

reg 为目标寄存器

描述:  $\text{reg} = \text{reg} \& \text{immediate}$

示例: `and_into 0xfb,pdata //pdata = pdata&0xfb`

## **iand:**

格式: `iand ,`

参数: regr 为源寄存器

regw 为目标寄存器

描述:  $\text{regw} = \text{pdata} \& \text{regr}$

示例: `iand regb,pdata //pdata = pdata & regb`

## **iand\_into:**

格式: `iand_into`

参数: reg 为目标寄存器

描述:  $\text{reg} = \text{pdata} \& \text{reg}$

示例: `iand_into reg//reg = pdata & regb`

备注: none



## **ior:**

格式: ior ,

参数: regr 为源数据地址

regw 为写入目标地址

描述: 从 regr 中读取数据与 pdata 取或操作, 并写入 regw

示例: ior regr,regw //regr 与 pdata 取或后存入 regw

备注: none

## **ixor:**

格式: ixor ,

参数: regr 为源数据地址

regw 为写入目标地址

描述: 从 regr 中读取数据与 pdata 取异或操作, 并写入 regw

示例: ixor rega,rega //rega 于 pdata 取异或然后存入  
rega

备注: none

## **or:**

格式: or ,,

参数: immediate 为立即数(最大为 9bit)

regr 为源寄存器

regw 为目标寄存器

描述:  $\text{regw} = \text{regr} | \text{immediate}$  按位或

示例: or clkn\_bt,0x1fc,bt\_clk //bt\_clk = clkn\_bt|0x1fc 按位或

备注:none

### **or\_into:**

格式: or\_into ,

参数: immediate 为立即数(最大为 9bit)

reg 为目标寄存器

描述:  $\text{reg} = \text{reg} \mid \text{immediate}$  按位或

示例: or\_into 0x1fc,bt\_clk //bt\_clk =bt\_clk | 0x1fc 按位或

备注:none

## **第四章运算指令**

### **add:**

格式: and ,,

参数: immediate 为立即数(最大为 9bit)

regr 为源寄存器

regw 为目标寄存器

描述:  $\text{regw} = \text{regr} + \text{immediate}$

示例: add clkn\_bt,0x1fc,bt\_clk //bt\_clk = clkn\_bt+0x1fc

备注:none

### **iadd:**

格式: iand ,

参数:

regr 为源寄存器

regw 为目标寄存器

描述:  $\text{regw} = \text{regr} + \text{pdata}$

示例: `iadd rega,rega //rega=rega+pdata`

## **increase:**

格式: `increase ,`

参数: `immediate` 为立即数(最大为 9bit)

reg 为目标寄存器

描述: 即  $\text{reg} = \text{immediate} + \text{reg}$

示例: `increase -1,pdata //pdata 减 1`

备注: none

## **pincrease:**

格式: `pincrease`

参数: `immediate` 为立即数(最大为 9bit)

描述: 即  $\text{pdata} = \text{immediate} + \text{pdata}$

示例: `pincrease 1 //pdata 加 1`

备注: none

## **sub:**

格式: `and ,,`

参数: `immediate` 为立即数(最大为 9bit)

regr 为源寄存器

regw 为目标寄存器

描述:  $\text{regw} = \text{immediate} - \text{regr}$

示例: sub pdata,4,null

备注:none

## **isub:**

格式: and ,

参数:

regr 为源寄存器

regw 为目标寄存器

描述:  $\text{regw} = \text{pdata} - \text{regr}$

示例: isub pdata,4,null

备注:none

## **imul32:**

格式: imul32

参数: regr 为源寄存器

描述: 32 位乘法器, 把 regr 与 pdata 相乘结果放在 pdata 中

示例: fetch 1,mem\_fcomp\_mul

hjam 0x04,rf\_pll\_rstn

imul32 temp,pdata //pdata=pdata\*temp

备注: none

## **mul32:**

格式: mul32 ,,

参数: regr 为源寄存器

regw 为目标寄存器

immediate 为立即数(最大为 9bit)

描述: 32 位乘法器, 把 regr 与 Immediate 相乘结果放在 regw 中

示例: fetch 1,mem\_select\_list\_item

```
mul32 pdata,7,pdata      //pdata 与 7 相乘结果放在
pdata 中
```

备注: none

## **div:**

格式: div ,

参数: regr 为源寄存器

immediate 为立即数(最大为 9bit)

描述: 除法器, 把 regr 与 Immediate 相除。

示例:

```
div pdata,10
```

```
call p_wait_div_end
```

.....

## **idiv:**

格式: div

参数: regr 为源寄存器

描述: 除法器, 把 pdata 与 regr 相除。

示例:

```
arg 0x10,pdata
```

```
arg 0x02,temp
```

```
idiv temp
```

```
call wait_div_end//注意, 除指令必须等待, 乘指令不需要
quotient pdata //商
```

remainder temp //余数

pdata /temp =16/2=8 余 0

.....

## **random:**

格式: random

参数: regw 为被置数的寄存器

描述: 将一个指定的寄存器 regw 置为随机数

示例: random pdata //将 pdata 置为一个随机数

备注:none

# 第五章位运算指令

## **lshift:**

格式: lshift ,

参数: regr 为源寄存器

regw 为目标寄存器

描述: 读取 regr 的数据, 并右移一位置入 regw, regr 不变

示例: lshift pdata,pdata //将 pdata 左移一位放入 pdata

备注: none

## **lshift2:**

格式: lshift2 ,

参数: regr 为源寄存器

regw 为目标寄存器

描述：读取 regr 的数据，并右移一位置入 regw, regr 不变

示例：lshift2 pdata, pdata      //将 pdata 左移 2 位放入 pdata

备注：none

### **lshift3:**

格式：lshift3 ,

参数：regr 为源寄存器

regw 为目标寄存器

描述：读取 regr 的数据，并右移一位置入 regw, regr 不变

示例：lshift3 pdata, pdata      //将 pdata 左移 3 位放入 pdata

备注：none

### **lshift4:**

格式：lshift4 ,

参数：regr 为源寄存器

regw 为目标寄存器

描述：读取 regr 的数据，并右移一位置入 regw, regr 不变

示例：lshift4 pdata, pdata      //将 pdata 左移 4 位放入 pdata

备注：none

### **rshift:**

格式：rshift ,

参数：regr 为源寄存器

regw 为目标寄存器

描述：从 regr 读取数据，并右移一位，置入 regw，regr 不变

示例：rshift pdata,pdata       //将 pdata 右移一位放入 pdata

备注：none

## **rshift2:**

格式：rshift2 ,

参数：regr 为源寄存器

regw 为目标寄存器

描述：从 regr 读取数据，并右移一位，置入 regw，regr 不变

示例：rshift2 pdata,pdata       //将 pdata 右移 2 位放入 pdata

备注：none

## **rshift3:**

格式：rshift4 ,

参数：regr 为源寄存器

regw 为目标寄存器

描述：从 regr 读取数据，并右移一位，置入 regw，regr 不变

示例：rshift3 pdata,pdata       //将 pdata 右移 3 位放入 pdata

备注：none

## **rshift4:**

格式：rshift4 ,

参数：regr 为源寄存器

regw 为目标寄存器

描述：从 regr 读取数据，并右移一位，置入 regw，regr 不变



示例: rshift4 pdata,pdata      //将 pdata 右移 4 位放入 pdata

备注: none

## **setflag:**

格式: setflag ,,

参数: flag      为源 falg 标记

Immediate 标记位(最大为 9bit)

reg      目标寄存器

描述: 将 flag 的值传给 reg 的 immediate

示例: setflag true,9,pdata      //将 true 的值赋值给 pdata 的第 9 位

备注: none

## **nsetflag:**

格式: nsetflag ,,

参数: flag 为源 falg 标记

immediate 标记位(最大为 9bit)

reg 目标寄存器

描述: 将 flag 的值取反传给 reg 的第 immediate 位

示例: nsetflag true,9,pdata      //将 true 的值取反后赋值给 pdata 的第 9 位

备注: none

## **isolate1:**

格式: isolate1 ,

参数: immediate 标记位(最大为 9bit)

reg 目标寄存器

描述：若 reg 的第 immediate 位的值为 1，则将 true 标志位置 1

示例：fetch 1,mem\_device\_option

isolate1 2,pdata //若 pdata 中的第 2 位为 1，则将 true 标志位置 1，否则为 0

call app\_ble\_start\_adv,true

备注：none

## **isolate0:**

格式：isolate0 ,

参数：immediate 标记位(最大为 9bit)

reg 目标寄存器

描述：若 reg 的第 immediate 位的值为 0，则将 true 标志位置 1

示例：fetch 1,mem\_device\_option

Isolate0 2,pdata //若 pdata 中的第 2 位为 0，则将 true 标志位置 1，否则为 0

call app\_ble\_start\_adv,true

备注：none

## **qisolate1:**

格式：qisolate1

参数：reg 目标寄存器

描述：若 reg 的第 queue 位的值为 1，则将 true 标志位置 1

示例：arg 3,queue

fetch 1,mem\_device\_option

qisolate1 pdata //若 pdata 中的第 3 位为 1，则将 true 标志位置 1，否则为 0

备注: none

## **qisolate0:**

格式: qisolate0

参数: reg 目标寄存器

描述: 若 reg 的第 queue 位的值为 0, 则将 true 标志位置 1

示例: arg 3,queue

```
fetch 1,mem_device_option
```

qisolate0 pdata //若 pdata 中的第 3 位为 0, 则将 true 标志位置 1 否则为 0

备注: none

## **setflip:**

格式: setflip ,

参数: Immediate 为标记位(最大为 9bit)

reg 为目标寄存器

描述: 将 reg 的 immediate 位取反

示例: setflip 3,pdata //将 pdata 的第 3 位取反

备注: none

## **set0:**

格式: set0 < immediate> ,

参数: immediate: 立即数, 需要清 0 的位(最大为 9bit); reg: 寄存器名, 目标寄存器。

描述: 将的第< immediate>位清 0。

示例: set0 2,pdata //将寄存器 pdata 中的第 2 位清 0。

备注: none

## **set1:**

格式: set1 < immediate> ,

参数: immediate: 立即数, 需要置 1 的位(最大为 9bit); reg: 寄存器名, 目标寄存器。

描述: 将的第< immediate>位置 1。

示例: set1 2,pdata

将寄存器 pdata 中的第 2 位置 1。

备注: none

## **qset0:**

格式: qset0

参数: 寄存器 queue 内的值为需要清 0 的位; reg: 目标寄存器。

描述: 将的第 queue 位清 0。

示例: arg 2,queue

setarg 0

qset0 pdata //将寄存器 pdata 中的第 2 位清 0。

备注: none

## **qset1:**

格式: qset1

参数: 寄存器 queue 内的值为需要清 1 的位; reg: 目标寄存器。

描述: 将的第 queue 位清 1。

示例: arg 2,queue

setarg 0

qset1 pdata //将寄存器 pdata 中的第 2 位清 1。

备注: none

## **byteswap:**

格式: byteswap

参数: regr 源寄存器

regw 目标寄存器

描述: 读取 regr 的值高位低为交换后, 在赋值给 regw

示例: byteswap regA, pdata //如 regA = 0x1112,则 pdata 为 0x1211

//如 regA = 0x45ff,则 pdata 为 0xff45

## **reverse:**

格式: reverse

参数: regr 源寄存器

regw 目标寄存器

描述: 读取 regr 的值取翻转, 在赋值给 regw

示例: reverse pdata, pdata //如 pdata 二进制 11001, 那么翻转后为 10011

//如 pdata 二进制 11000111, 那么翻转后为 11100011

备注: none

## **invert:**

格式: invert ,

参数: regr 源寄存器

regw 目标寄存器

描述：读取 `regw` 的值存入 `regw`，在将 `regw` 取反码

示例：`invert pdata,pdata`            `//pdata = ~pdata`

备注：none

## **compare:**

格式：`compare < immediate>,,`

参数：Immediate 参数 1(最大为 9bit)

reg 参数 2

mask 参数 3

描述：在 `immediate` 与 `reg` 相同时，设置 `cond flag (flag true)`，  
`mask` 标记比较的位数，如 `mask` 为 `0xff` 比较低八位，`mask` 为 `0x0f` 比较低四位。

示例：`compare 0x3a,pdata,0xff` //比较 3a 与 `pdata` 的低八位，若相同  
则将 `true` 置为 1

`nrt true`            //判断 `true`，若为 0 则返回，若为 1，则继续执行

备注：none

## **icompare:**

格式：`icompare ,`

参数：reg 参数 1

mask 参数 2

描述：在 `pdata` 与 `reg` 相同时，置 `cond flag (flag true)` 为 0，  
`mask` 标记比较的位数，如 `mask` 为 `0xff` 比较低八位，`mask` 为 `0x0f` 比较低四位。

示例：`icompare 0xff,temp`            //比较 `pdata` 与 `temp` 的低八位

`branch process_acl,true`

备注：none

## 第六章寻址指令

### jam:

格式: jam ,

参数: immediate 为立即数(最大为 8bit)

addr 为目标地址

描述: 将立即数 immediate 载入到地址 addr

示例: jam 0x01,mem\_fw\_ver

jam 0,mem\_hci\_cmd

备注: jam 的寻址范围是从 0x00 开始的内存。

### hjam:

格式: hjam ,

参数: immediate 为立即数(最大为 8bit)

addr 为目标地址

描述: 将立即数 immediate 载入到地址 addr

示例: hjam 0,core\_config //将 core\_config 置为 0

备注: hjam 的寻址范围是从 0x8000 开始的硬件寄存器

### fetch:

格式: fetch ,< addr>

参数: num\_bytes 为读取数据的字节数

addr 为源地址

描述：从 `addr` 中读取 `num_bytes` 的数据存入 `pdata`

示例：`fetch 1,0x5` //从内存 `0x5` 中读取 1 字节数据存入 `pdata` 中。

备注：该指令会影响 `blank` 标志位。

## **fetcht:**

格式：`fetcht ,< addr>`

参数： `num_bytes` 为读取数据的字节数

`addr` 为源地址

描述：从 `addr` 中读取 `num_bytes` 的数据存入 `temp`

示例：`fetcht 1,0x5` //从内存 `0x5` 中读取 1 字节数据存入 `temp` 中。

备注：none

## **ifetch:**

格式：`ifetch , < reg>`

参数： `num_bytes` 为读取数据的字节数

`reg` 为存储源地址的寄存器

描述：从 `reg` 存储的地址中读取 `num_bytes` 的数据存入 `pdata`

示例： `ifetch 1,contr` //从 `contr` 存储的地址中读取 1 字节的数据存入 `pdata`

备注： `ifetch` 的操作为间接寻址操作。该操作会影响 `blank` 标志位。

## **ifetcht:**

格式： `ifetcht , < reg>`

参数： `num_bytes` 为读取数据的字节数

`reg` 为存储源地址的寄存器



描述：从 `reg` 存储的地址中读取 `num_bytes` 的数据存入 `temp`

示例：`ifetch 1,contr` //从 `contr` 存储的地址中读取 1 字节的数据存入 `pdata`

备注：`ifetch` 的操作为间接寻址操作。

## **hfetch:**

格式：`hfetch` , `< reg>`

参数：`num_bytes` 为读取数据的字节数

`reg` 为存储源地址的硬件寄存器

描述：从 `reg` 存储的地址中读取 `num_bytes` 的数据存入 `pdata`

示例：`hfetch 1,0x811c`//从 `0x811c` 存储的地址中读取 1 字节的数据存入 `pdata`

备注：`hfetch` 的寻址范围是从 `0x8000` 开始的硬件寄存器，该操作会影响 `blank` 标志位。

## **hfetcht:**

格式：`hfetcht` , `< reg>`

参数：`num_bytes` 为读取数据的字节数

`reg` 为存储源地址的寄存器

描述：从 `reg` 存储的地址中读取 `num_bytes` 的数据存入 `temp`

示例：`hfetcht 1,0x811c` //从 `0x811c` 存储的地址中读取 1 字节的数据存入 `temp`

备注：`hfetcht` 的寻址范围是从 `0x8000` 开始的硬件寄存器，该操作会影响 `blank` 标志位。

## **store:**

格式: store ,

参数: num\_bytes 为读取数据的字节数

addr 为写入数据的地址

描述: 从 pdata 中读取指定长度 num\_byte 字节数据写入指定地址 addr。

示例: store 3,0x0 //从 pdata 读取 3 个字节的数据存入起始地址为 0x0。

备注: store 的寻址范围是 0x0000 地址为起始的内存。

## **storet:**

格式: storet ,

参数: num\_bytes 为读取数据的字节数

addr 为写入数据的地址

描述: 从 pdata 中读取指定长度 num\_byte 字节数据写入指定地址 addr。

示例: storet 3,0x0 //从 pdata 读取 3 个字节的数据存入起始地址为 0x0。

备注: storet 的寻址范围是 0x0000 地址为起始的内存。

## **istore:**

格式: istore ,

参数: num\_bytes 为读取数据的字节数

reg 为间接寻址寄存器

描述: 从 pdata 中读取指定长度数据写入指定寄存器存有的地址

示例: istore 2,contw //从 pdata 读取 2 字节的数据存入 contw

备注: **istore** 的操作为间接寻址操作。

## **istoret:**

格式: **hstore** ,

参数: **num\_bytes** 为读取数据的字节数

**reg** 为存储目标内存地址的寄存器

描述: 从 **temp** 中读取指定长度数据写入指定寄存器

示例: **istoret 1,contw** //从 **temp** 读取 1 个字节的数据写入 **contw**

备注: **istoret** 的操作为间接寻址操作。

## **hstore:**

格式: **hstore** ,

参数: **num\_bytes** 为读取数据的字节数

**addr** 为写入数据的地址

描述: 从 **pdata** 中读取指定长度 **num\_byte** 字节数据写入指定地址 **addr**

示例: **hstore 2,0x8848** //从 **pdata** 读取 2 个字节的数据存入 **0x8848**

备注: **hstore** 的寻址范围是 **0x8000** 地址为起始的硬件寄存器

## **force:**

格式: **force** ,

参数: **Immediate** 为立即数(最大为 9bit), **regs** 为目标寄存器

描述: 将立即数输入到目标寄存器中保存。

示例: **force 0,pdata**//**force 0x0 into pdata**

备注: 请勿使用此指令, 为代码统一, 寄存器赋值采用 **arg** 指令

## **iforce:**

格式: iforce

参数: regw 为目标寄存器

描述: 把 pdata 的数据赋值给 regw

示例: iforce am\_addr //将 pdata 的数据赋值给寄存器 am\_addr。

备注: 请勿使用此指令, 为代码统一, 寄存器拷贝使用 copy 指令

## **arg:**

格式: arg ,

参数: immediate 立即数(最大为 21bit)

reg 目标寄存器

描述: 将立即数输入到指定寄存器 reg。

示例: arg 0x0,contw //put 0x0 Into contw

arg 0x400,loopcnt //put 0x400 into loopcnt

备注: none

## **setarg:**

格式: setarg

参数: Immediate 为立即数(最大为 27bit)

描述: 将立即数送入 pdata

示例: setarg 0x4f9 // 将 0x4f9 放入 pdata

备注: none

## **copy :**

格式: copy ,

参数: regr 为源寄存器

regw 目标寄存器

描述: 通过将 regr 值复制到 regw 中去

示例: copy rega,alarm //将 rega 的值赋值给 alarm

备注: none

## **deposit:**

格式: deposit

参数: reg 为目标寄存器

描述: 通过 alu 将 reg 赋值给 pdata

示例: deposit clkn\_bt //将 clkn\_bt 的值赋值给 pdata

备注: 请勿使用此指令, 为代码统一, 寄存器拷贝使用 copy 指令

## **disable:**

格式: disable

参数: flag 为目标标记位

描述: 将 flag 置为 0

示例: disable slave2 //将 slave2 标记为 0

备注: none

## **enable:**

格式: enable

参数: flag 为目标标记位

描述: 将 flag 置为 1

示例: enable slave2 //将 slave2 标记为 1

备注: none

# 第七章跳转指令

## call:

格式: `call` ,

参数: `func` 为目标地址

`flag` 为判据,缺省时即强制跳转

描述: 跳转到指定标记, 并保存程序当前地址, 用 `rtn` 返回。相当于调用函数的功能。区别于 `branch` 系列指令, `branch` 仅仅跳转不能用 `rtn` 返回

示例: `call clean_mem //jmp to clean_mem 函数中`

备注: none

## rtn:

格式: `rtn`

参数: `flag` 为判断依据

描述: 与 `call` 相对应, 返回到 `call` 调用位置, 并继续执行

示例:

`eeeprom_load_reconn_info:`

.....

`rtn`

备注: 区别于 `nrtn`

## nrtn:

格式: `nrtn`

参数: `flag` 为判断依据

描述: `flag` 为 0 则返回, 若 `flag` 为 1 则不返回。

## **rtneq:**

格式: rtneq

参数: immediate 为指定数值(最大为 8bit)。

描述: 若寄存器 pdata 的数值与 immediate 值相等则返回, 否则继续执行。

示例: fetch 1,mem\_sp\_calc

rtnbit1 7 //判断寄存器 pdata 的第七位是否为 1,若为 1 则返回, 否则继续执行。

备注: none

## **rtnbit1:**

格式: rtnbit1

参数: immediate 为指定寄存器 pdata 的位(最大为 8bit)。

描述: 若 pdata 的第 immediate 位为 1 则将返回, 否则继续执行。

示例: fetch 1,mem\_sp\_calc

rtnbit1 7 //判断寄存器 pdata 的第七位是否为 1,若为 1 则返回, 否则继续执行。

备注: none

## **rtnbit0:**

格式: rtnbit0

参数: immediate 为指定寄存器 pdata 的位(最大为 8bit)。

描述: 若 pdata 的第 immediate 位为 0 则将返回, 否则继续执行。

示例: fetch 1,mem\_sp\_calc

Rtnbit0 7 //判断寄存器 pdata 的第七位是否为 0, 若为 0 则返回, 否则继续执行。

备注: none

## **branch:**

格式: branch ,

参数: addr 为跳转目标地址

flag 为跳转判据仅当 flag 为 1 时候发生跳转至地址 addr

描述: branch

addr 为跳转目标地址,强制跳转至地址 addr

示例: branch eckp\_calc\_init\_1,true //当 flag true 为 1 时候发生跳转

branch ec\_copy //强制跳转到

## **nbranch:**

格式: nbranch ,

参数: addr 为跳转目标地址

flag 为跳转判据

描述: 仅当 flag 为 0 时候发生跳转至地址 addr

示例: nbranch eckp\_calc\_init\_1,true //当 flag true 为 0 时候发生跳转

branch ec\_copy //强制跳转到

备注: none

## **bbit1:**

格式: bbit1 ,< addr>

参数: immediate 为指定 pdata 位的地址(最大为 8bit)

addr 为跳转地址

描述: 如果 pdata 的第 immediate 位为 1, 则将跳转到 addr 继续执行



示例: `bbit1 3,init_uuid_kb` //如果 `pdata` 的第 3 位为 1, 则跳转到 `init_uuid_kb`

备注: none

## **bbit0:**

格式: `bbit0 ,< addr>`

参数: `immediate` 为指定 `pdata` 位的地址(最大为 8bit)

`addr` 为跳转地址

描述: 如果 `pdata` 的第 `immediate` 位为 0, 则将跳转到 `addr` 继续执行

示例: `bbit0 3,init_uuid_kb` //如果 `pdata` 的第 3 位为 0, 则跳转到 `init_uuid_kb`

备注: none

## **beq:**

格式: `beq ,`

参数: `imme` 为立即数(最大为 8bit)

`addr` 为跳转地址

描述: 若 `imme` 与 `pdata` 相等,则跳转到 `addr`

示例: `beq 1,setup_clk` //若 `pdata==1`, 则跳转到 `setup_clk` 继续执行

备注: none

## **bne:**

格式: `bne ,`

参数: `imme` 为立即数(最大为 8bit)

`addr` 为跳转地址

描述：若 `imme` 与 `pdata` 不相等,则跳转到 `addr`

示例：`bne 1,setup_clk`     //若 `pdata != 1`，则跳转到 `setup_clk` 继续执行

备注：none

## **loop:**

格式：loop

参数：addr    跳转目标地址

描述：如果 `loopcnt` 不为 0 则跳转到 `addr`，每次跳转后 `loopcnt - 1`

示例：loop pn9\_loop                 //如果 `loopcnt` 不为 0 则跳转到 `pn9_loop`

备注：none

## **bpatch:**

格式：bpatch ,

参数：Immediate 立即数(最大为 8bit)

addr 目标地址

描述：该指令为代码打 patch 使用的，若 `addr` 的第 Immediate 位为 1 跳转到 patch 函数

示例：

bpatch patch01\_0,mem\_patch01//若变量 `mem_patch01` 的第 0 个 bit 为 1 则跳转 patch 中

备注：none

## **parse:**

格式：parse ,,

参数：source 为数据源(硬件数据源)

dest 为数据目标存储区域（硬件地址）

immediate 为传输的 bit(最大为 9bit)

描述：把数据源 source 的 immediate 个 bit 的数据送到目的地址 pwindow

示例：parse demod,bucket,8 //从 demod 中取一个字节 8 个 bit

rshift3 pwindow,pdata

store 1,mem\_le\_rxbuf //将一个字节存入 mem\_le\_rxbuf

备注：none

## 第八章数据类型

本章主要是讲述汇编在使用时的数据类型。

### 常量

在程序运行过程中，其值不能改变的量称为常量。如示例：jam 0,mem\_le\_adv\_enable、

setarg 10、arg 5,temp 中的数据 0、10、5 就是常量。数值常量就是数学中的常数，在程序中经常是给变量做赋值使用。

### 变量

如示例中 jam 0,mem\_hci\_cmd、

jam BT\_CMD\_STORE\_RECONN\_INFO\_LE,mem\_fifo\_temp、hjam 5,core\_gpio\_out1 的 mem\_hci\_cmd、mem\_fifo\_temp 和

core\_gpio\_out1 是变量。其中，

BT\_CMD\_STORE\_RECONN\_INFO\_LE 为一个宏，实质代表一个固

定的常量，只是为了方便阅读识别。变量是代表一个有名字的、具有特定属性的一个存储单元。它用来存放数据，也就是存放变量的值，在程序运行期间变量的值是可以改变的。

变量必须先定义后使用，在定义时指定该变量的名字和大小，一个变量应该有一个名字

以便被使用，变量的定义是在`.format`文件中定义的。如示例 2 `mem_mouse_x` 中 2 为变量的大小，`mem_mouse_x` 为变量的名字，定义了名字以后，代码在编译时会给 `mem_mouse_x` 这个变量分配一个内存地址，作为数据存储的真正位置。使用时只要对该名字变量进行赋值、取值操作即可。示例为 `fetch 2,mem_mouse_x`、`store 2,mem_mouse_x`。但要注意在取值、赋值操作时数据的大小不要超越变量申请时的大小，特殊情况除外（代码优化）。

## 第九章汇编语句

### 汇编语句的作用和分类

在程序代码中我们可以看到很多执行的函数，而每个函数又是由众多语句组成的，有的只含有一个语句，有的含有多个语句。语句的作用是让 **CPU** 执行我们想要的操作。

汇编语句分为以下几类：

- ①条件语句
- ②循环语句
- ③跳转语句
- ④多支条件语句

## ⑤从函数返回语句

由以上几种结构构成了丰富多样，功能各异的函数。

## 条件语句

条件语句顾名思义就是当条件满足时就要执行某种操作，例如 C 语言中 if、if else 语句所实现的功能。

编写程序：

mouse\_send\_process:

```
    fetch 1,mem_app_handshake_flag
    rtn blank//①
    call l2cap_malloc_is_fifo_empty
    nrtn blank//②
    call mouse_clear_sensor
    call mouse_motion
    nbranch mouse_clean_payload,user//③
    fetch 1,mem_mouse_move_flag
    branch mouse_clean_sensor_init,blank //④
```

.....

程序分析：上面为一段条件语句的小程序，其中 mouse\_send\_process:为函数名称标签。①在变量 mem\_app\_handshake\_flag 取值，通过 blank 标志位来判断条件是否满足，当条件满足 blank 为 1 时，则会执行 rtn 操作，如果 blank 为 0 条件则继续向下执行。②程序中的 nrtn blank 恰恰与第一个相反的，当 blank 为 0 时执行 rtn 操作，当 blank 为 1 时向下继续执行。③通过 user 标志位来作为判断的条件，当 user 为 0 时则执行跳转到函数 mouse\_clean\_payload 中，当 user 为 1 时则继续向下执行。④通过 blank 来判断的，当 blank 为 1 时跳转到函数

mouse\_clean\_sensor\_init 中，当 blank 为 0 时则继续执行。在汇编中可以通过 rtn、branch 等指令来完成条件判断语句的。

判断语句的结构为：

；

如果条件满足，执行该指令，不满足则忽略。

## 多支条件语句

上面介绍的条件语句只有两个条件条件可以选择，但在实际的问题中常常需要用到多分支的选择。比如学生的成绩为 95 分以上、85 分以上、75 分以上、60 分以上等。当然，可以使用多个条件语句进行多次判断来完成，但我们提供了更方便的多支条件语句来实现。

beq 可以实现多分支条件语句的判断。

编写程序：

```
fetcht 1,mem_mouse_z_last
    iadd temp,pdata
    beq 0x38,mouse_wheel_forward
    beq 0x34,mouse_wheel_back
    beq 0x0b,mouse_wheel_back
    beq 0x07,mouse_wheel_forward
    rtn
```

程序分析：将运算后的 pdata 寄存器数值做判断处理，若等于 0x38 则跳转到函数 mouse\_wheel\_forward；若等于 0x34 则跳转到函数 mouse\_wheel\_back；若等于 0x0b 则跳转到函数 mouse\_wheel\_back；若等于 0x07 则跳转到函数 mouse\_wheel\_forward。

bbit1/bbit0 也可实现多个分支条件语句的判断。

编写程序：

```
fetch 1,mem_device_option
bbit1 6,mouse_init
bbit1 7,kb_init
rtn
```

程序分析：根据 `pdata` 寄存器中的数值来做判断处理，若第 6 位为 1 则跳转到 `mouse_init` 函数中，若第 7 位为 1 则跳转到 `kb_init` 函数。

## 循环语句

前面介绍了条件选择结构，但在生活中或是在程序的所处理的问题中常常遇到需要重复处理的问题。例如：

要读取内存数据 50 次；

要写内存数据 50 次；

要处理这类问题时，如果以最原始的方法就是，写 50 遍相同的程序。但这样的处理太过于繁琐，所以为了效率高效我们加入了循环结构。

使用 `loop` 实现循环结构：

编写程序：

```
arg 8,loopcnt
arg core_usb_dfifo1,rega
arg 1,queue
usb_tx_loop:
    ifetch 1,contr
    istore 1,rega
    loop usb_tx_loop
```

程序分析：首先，通过 `arg` 和 `loopcnt` 寄存器输入想要循环的次数，后通过 `loop` 来实现跳转，每跳转一次 `loopcnt` 寄存器会自动减 1，直到 `loopcnt` 寄存器为 0 才继续向下执行。

通过 `branch` 实现循环等待功能：

`mouse_init_p3204:`

```
    setarg 0
    call twspi_read
    store 1,mem_sensor_id
    beq P3204_ID,mouse_init_p3204_cont
    call twspi_reset
    nop 10000
    branch mouse_init_p3204
```

程序分析：该段程序是通过 `branch` 完成一个死循环功能，每次会判断 `mem_sensor_id` 的值是否等于 `P3204_ID`，如果等于才会跳出该循环中，否则一直循环。