YC1021

易兆 YiChip 蓝牙 ASM 手册

Yichip Microelectronics

Revision History

Version	Date	Author	Description
V1.0	2014.9.14	邵怡迪	Initial version
V1.1	2014.9.15	邵怡迪	添加常用指令 org 起
V1.2	2014.9.16	邵怡迪	添加常用命令 compare 起，更改格式
V1.3	2014.9.18	邵怡迪	修改排版，增加指令
V1.4	2014.9.24	邵怡迪	增加指令
V1.5	2014.9.26	寇凡	修改排版，增改寄存器说明
V1.6	2014.9.29	张胜利	修改排版，增改常用指令描述、示例
V1.7	2015.10.28	王洋	修改常用指令，添汇编语句，数据类型
V1.8	2016.10.27	牟起航	添加一些常用指令，修改排版，修改描述错误
V1.9	2016.11.18	牟起航	添加 clkn 和 clke 寄存器的描述
V1.10	2016.12.07	牟起航	添加 alarm、regext、stop_watch、timeup 寄存器，修改一下错误描述。增加除法的取商和取余的指令。

目录

1	add	44	iflip	87	qisolate1
2	aligned	45	iforce	88	qset0
3	and	46	iinject	89	qset1
4	and_into	47	imul32	90	qsetflag
5	arg	48	imult	91	qsetflip
6	bbit0	49	increase	92	quotient
7	bbit1	50	inject	93	random
8	beq	51	invert	94	regexrot
9	bmark0	52	ior	95	remainder
10	bmark1	53	ior_into	96	reverse
11	bne	54	isolate0	97	revision
12	bpatch	55	isolate1	98	rshift
13	branch	56	istore	99	rshift16
14	byteswap	57	istoret	100	rshift2
15	call	58	isub	101	rshift3
16	clear_stack	59	iverify	102	rshift32
17	compare	60	ixor	103	rshift4
18	copy	61	jam	104	rshift8
19	correlate	62	loop	105	rtn
20	crc	63	lshift	106	rtnbit0
21	deposit	64	lshift16	107	rtnbit1
22	disable	65	lshift2	108	rtneq
23	div	66	lshift3	109	rtnmark0
24	enable	67	lshift4	110	rtnmark1
25	fetch	68	lshift8	111	rtnne
26	fetcht	69	mul32	112	set0
27	flip	70	mult	113	set1
28	force	71	nbranch	114	setarg
29	hfetch	72	ncall	115	setflag
30	hfetcht	73	nop	116	setflip
31	hjam	74	nqsetflag	117	setsect
32	hstore	75	nrtn	118	shasx
33	hstoret	76	nsetflag	119	skip_to
34	iadd	77	or	120	sleep
35	ialigned	78	or_into	121	snooze
36	iand	79	parse	122	store
37	iand_into	80	pincrease	123	storet
38	icompare	81	pinvert	124	stuff
39	icopy	82	preload	125	sub
40	icrc	83	priority	126	until
41	idiv	84	product	127	xor
42	ifetch	85	pulse	128	xor_into
43	ifetcht	86	qisolate0		

第一章 寄存器

pdata：

属性：RW
说明：通用寄存器，64bits。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判断的。
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，64bits
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 1,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。

alarm:

属性：RW
说明：通用寄存器，44bits
备注：none
示例：

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

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

regext:

属性：RW
说明：通用寄存器，32bits
备注：内有 12 个寄存器，用 regext_index 作为索引访问。当 regext_index 等于 1 时选中 regext1，当 regext_index 等于 2 时选中 regext2。
示例：

```
arg 1,regext_index
arg 12,regext
arg 2,regext_index
arg 13,regext
copy regext,temp    //temp 的值为 13
arg 1,regext_index
copy regext,pdata //pdata 的值为 12
...
```

stop_watch:

属性：RW
说明：时间寄存器，20bits
备注：enable swfine 时每 clock 自动减 1, disable swfine 时每 625us 自动减 1 直到 0。可作为普通寄存器使用。

timeup:

属性：RW
说明：通用寄存器，28bits
备注：普通寄存器使用。

queue:

属性: RW
说明：通用寄存器，8bits
备注：一般搭配 qset,qisolate1 等指令使用，也可作为普通寄存器使用。
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
......
```

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。jam，hjam 等指令会影响该寄存器的值，在 jam，hjam 指令之后不要再使用该寄存器内的值。
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：当打开 uart 复用功能时，ifetch 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：当打开 uart 复用功能时，istore 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 32 bits，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。当 loopcnt 等于 0 时，loop 跳转会将 loopcnt 减成 0xffff，loopcnt 初始值不能为 0。
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：不建议使用，否则某些标志位可能会出问题。
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。
    bmark0 11,dsc_string2_short //判断 Mark 寄存器的第 11 位是否为 0，若为 0 则跳转到 dsc_string2_short 函数中
    ......
```

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:

属性：RW
说明：时间寄存器，16bits + 28bits。该寄存器会随着芯片 runing 不断自增。每一个 clock 时间增加 1，clock 的晶振的频率有关，12MHz 晶振的 clock 为 1/12us
备注：蓝牙连接时，master 的时钟，slave 以该时钟为校正时钟。
示例：

```
deposit clkn          //将 clkn 内时间数值取出放入 pdata
store 6,mem_clkn
.....
```

clkn_bt:

属性：RW
说明：时间寄存器，28bits。会随着芯片 runing 不断自增。达到最大值则从零开始。该寄存器仅仅为 clkn 寄存器的高 28bits。
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，16bits。会随着芯片 runing 不断自增。达到最大值则从零开始。该寄存器仅仅为 clkn 寄存器的低 16bits。
备注：与 clkn_bt 精度不同，clkn_rt 精度更高。clkn_rt 默认的累加范围为 0~0xea6，clkn_rt 溢出之后，clkn_bt 会累加 1.
示例：

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

.....

clke:

属性：RW
说明：时间寄存器，16bits + 28bits。该寄存器会随着芯片 runing 不断自增。每一个 clock 时间增加 1，clock 的晶振的频率有关，12MHz 晶振的 clock 为 1/12us
备注：蓝牙连接时，slave 的时钟，根据 master 的时钟需要设置。
示例：

```
deposit clke          //将 clkn 内时间数值取出放入 pdata
store 6,mem_clkn
.....
```

clke_bt:

属性：R
说明：时间寄存器，28bits。会随着芯片 runing 不断自增。达到最大值则从零开始。该寄存器仅仅为 clke 寄存器的高 28bits。
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。该寄存器仅仅为 clke 寄存器的低 16bits。
备注：与 clke_bt 精度不同，clke_rt 精度更高。clke_rt 默认的累加范围为 0~0xea6，clkn_rt 溢出之后，clke_bt 会累加 1
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 le_mouse.
.....
```

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、ifetch、setarg 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
rtn 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。isub 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、isolate1 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 2，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 3，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，以实现对 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 <regr>,<immediate>,<regw>
参数：immediate 为立即数
regr 为源寄存器
regw 为目标寄存器
描述：regw = regr&immediate
示例：and clkn_bt,0x1fc,bt_clk //bt_clk = clkn_bt&0x1fc
备注:none

and_into：

格式：and_into <immediate> ,<reg>
参数：immediate 为立即数
reg 为目标寄存器
描述：reg = reg&immediate
示例：and_into 0xfb,pdata //pdata = pdata&0xfb

iand：

格式：iand <regr> ,<regw>
参数：regr 为源寄存器
regw 为目标寄存器
描述：regw = pdata & regr
示例：iand regb,pdata //pdata = pdata & regb

iand_into：

格式：iand_into <regw>
参数：regw 为目标寄存器
描述：regw = pdata & regw
示例：iand_into regb //regb = pdata & regb
备注：none

ior:

格式：ior <regr>,<regw>
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 <regr>,<regw>
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 与 pdata 取异或然后存入 rega
备注：none

or:

格式：or <regr>,<immediate>,<regw>
参数：immediate 为立即数
regr 为源寄存器
regw 为目标寄存器
描述：regw = regr|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 <immediate>,<reg>
参数：immediate 为立即数
reg 为目标寄存器
描述：reg= reg | immediate 按位或
示例：or_into 0x1fc,bt_clk //bt_clk =bt_clk | 0x1fc 按位或
备注:none

nop:

格式 nop <immediate>
参数：immediate 为立即数
描述：空指令，主要用于延时，单位为 1/12 us
示例：

```
nop 1000    //延时 1000*1/12 us
```

备注:none

第四章 运算指令

add:

格式：and <regr>,<immediate>,<regw>
参数：immediate 为立即数
regr 为源寄存器
regw 为目标寄存器
描述：regw = regr+immediate
示例：add clkn_bt,0x1fc,bt_clk //bt_clk = clkn_bt+0x1fc
备注:none

iadd:

格式：iand <regr>,<regw>
参数:
regr 为源寄存器
regw 为目标寄存器
描述：regw = regr+pdata
示例：iadd rega,rega //rega=rega+pdata

increase:

格式：increase <immediate>,<reg>
参数：immediate 为立即数
reg 为目标急寄存器
描述：即 reg = immediate + reg
示例:increase -1,pdata //pdata 减 1
备注：none

pincrease:

格式：pimcrease <immediate>
参数：immediate 为立即数
描述：即 pdata = immediate + pdata
示例：pimcrease 1 //pdata 加 1
备注：none

sub:

格式：and <regr>,<immediate>,<regw>
参数：immediate 为立即数
regr 为源寄存器
regw 为目标寄存器
描述：regw = immediate-regr
示例：sub pdata,4,null
备注:none

isub:

格式：and <regr>,<regw>
参数：
regr 为源寄存器
regw 为目标寄存器
描述：regw = pdata-regr
示例：isub pdata,null
备注:none

imul32：

格式： imul32 <regr>,<regw>
参数： regr 为源寄存器
regw 为目标寄存器
描述：32 位乘法器，把 regr 与 pdata 相乘结果放在 regw 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 <regr>,<immediate>,<regw>
参数： regr 为源寄存器
regw 为目标寄存器
immediate 为立即数
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 <regr>,<immediate>
参数： regr 为源寄存器
immediate 为立即数
描述：除法器，把 regr 与 Immediate 相除。
示例：

```
div pdata,10
call wait_div_end
.....
```

idiv：

格式： div <regr>
参数： regr 为源寄存器
描述：除法器，把 pdata 与 regr 相除。
示例：

```
fetcht 2,mem_tsniff
deposit rega
idiv temp
call wait_div_end
.....
```

quotient：

格式： quotient<regr>
参数： regr 为源寄存器
描述：取商，将 div 或者 idiv 指令之后，将商取出存在 regr 寄存器中
示例：

```
setarg 10
div pdata,3
call wait_div_end   //等待除法完成
quotient temp       //temp 的值为 3
...
```

remainder：

格式： remainder<regr>
参数： regr 为源寄存器

描述：取余，将 div 或者 idiv 指令之后，将余数取出存在 regr 寄存器中
示例：

```
setarg 10
div pdata,3
call wait_div_end   //等待除法完成
remainder temp      //temp 的值为 1
...
```

random：

格式：random <regw>
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 <regr>,<regw>
参数： regr 为源寄存器
regw 为目标寄存器
描述：读取 regr 的数据，并左移一位置入 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 <regr>,<regw>
参数： regr 为源寄存器
regw 为目标寄存器
描述：读取 regr 的数据，并左移两位置入 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 <regr>,<regw>
参数： regr 为源寄存器
regw 为目标寄存器
描述：读取 regr 的数据，并左移三位置入 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 <regr>,<regw>
参数： regr 为源寄存器
regw 为目标寄存器
描述：读取 regr 的数据，并左移四位置入 regw,regr 不变
示例：lshift4 pdata,pdata //将 pdata 左移 4 位放入 pdata
备注：none

lshift8：

格式： lshift8 <regr>,<regw>
参数： regr 为源寄存器
regw 为目标寄存器
描述：读取 regr 的数据，并左移八位置入 regw,regr 不变
示例：lshift8 pdata,pdata //将 pdata 左移 8 位放入 pdata
备注：none

lshift16：

格式： lshift16 <regr>,<regw>
参数： regr 为源寄存器
regw 为目标寄存器
描述：读取 regr 的数据，并左移十六位置入 regw,regr 不变
示例：lshift16 pdata,pdata //将 pdata 左移 16 位放入 pdata
备注：none

rshift：

格式： rshift <regr>,<regw>
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 1 位放入 pdata
备注：none

rshift2：

格式：rshift2 <regr>,<regw>
参数： regr 为源寄存器
regw 为目标寄存器
描述：从 regr 读取数据，并右移两位，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： rshift3 <regr>,<regw>
参数： regr 为源寄存器
regw 为目标寄存器
描述：从 regr 读取数据，并右移三位，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 <regr>,<regw>
参数： regr 为源寄存器
regw 为目标寄存器
描述：从 regr 读取数据，并右移四位，置入 regw，regr 不变
示例：rshift4 pdata,pdata //将 pdata 右移 4 位放入 pdata
备注：none

rshift8：

格式： rshift8 <regr>,<regw>
参数： regr 为源寄存器
regw 为目标寄存器
描述：从 regr 读取数据，并右移八位，置入 regw，regr 不变
示例：rshift8 pdata,pdata //将 pdata 右移 8 位放入 pdata
备注：none

rshift16：

格式：rshift16 <regr>,<regw>
参数： regr 为源寄存器
regw 为目标寄存器
描述：从 regr 读取数据，并右移十六位，置入 regw，regr 不变
示例：rshift16 pdata,pdata //将 pdata 右移 16 位放入 pdata
备注：none

rshift32：

格式： rshift32 <regr>,<regw>
参数： regr 为源寄存器
regw 为目标寄存器
描述：从 regr 读取数据，并右移三十二位，置入 regw，regr 不变
示例：rshift32 pdata,pdata //将 pdata 右移 32 位放入 pdata
备注：none

setflag：

格式：setflag <flag>,<immediate>,<reg>
参数： flag 为源 flag 标记
Immediate 标记位
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

qsetflag

格式：qsetflag <flag>,<reg>
参数： flag 为源 flag 标记
reg 目标寄存器
描述：将 flag 的值传给 reg 的第 queue 位
示例: qsetflag true,pdata //将 true 的值赋值给 pdata 的第 queue 位
备注：none

nsetflag：

格式：nsetflag <flag>,<immediate>,<reg>
参数： flag 为源 flag 标记
immediate 标记位
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

nqsetflag

格式：nqsetflag <flag>,<reg>
参数： flag 为源 flag 标记
reg 目标寄存器
描述：将 flag 的值取反传给 reg 的第 queue 位
示例: nqsetflag true,pdata //将 true 的值取反后赋值给 pdata 的第 queue 位
备注：none

isolate1：

格式：isolate1 <immediate>,<reg>
参数：immediate 标记位
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 <immediate>,<reg>
参数：immediate 标记位
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 <reg>
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 <reg>
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 <immediate>, <reg>
参数：Immediate 为标记位
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>,<reg>
参数：immediate：立即数，需要清 0 的位；reg：寄存器名，目标寄存器。
描述：将<reg>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>,<reg>
参数：immediate：立即数，需要置 1 的位；reg：寄存器名，目标寄存器。
描述：将<reg>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 <reg>
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<reg>的第 queue 位清 0。
示例：arg 2,queue
setarg 0xff
qset0 pdata //将寄存器 pdata 中的第 2 位清 0。
备注：none

qset1：

格式：qset1 <reg>
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<reg>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 <regr><regw>
参数：regr 源寄存器
regw 目标寄存器
描述：读取 regr 的值高低字节为交换后，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 <regr><regw>
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 <regr>,<regw>
参数：regr　源寄存器
　　regw　目标寄存器
描述：读取 regr 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>,<reg>,<mask>
参数：Immediate 参数 1
　　reg 参数 2
　　mask 参数 3
描述：在 immediate 与 reg 相同时候，设置 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
　　nrtn 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 <mask>,<reg>
参数：reg 参数 1
　　mask 参数 2
描述：在 pdata 与 reg 相同时候，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：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 <immediate>,<addr>
参数：immediate 为立即数
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，该指令会影响 contw 寄存器的值。

hjam：

格式：hjam <immediate>,<addr>
参数：immediate 为立即数
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，该指令会影响 contw 寄存器的值。

fetch：

格式：fetch <num_bytes>,< 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，该操作会影响 contr 寄存器的值。

fetcht：

格式：fetcht <num_bytes>,< 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：该操作会影响 contr 寄存器的值

ifetch：

格式：ifetch <num_bytes>, < 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，该操作会影响 contr 寄存器的值。

ifetcht：

格式：ifetch <num_bytes>, < 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: ifetcht 1,contr //从 contr 存储的地址中读取 1 字节的数据存入 pdata
备注：ifetcht 的操作为间接寻址操作，该操作会影响 contr 寄存器的值。

hfetch：

格式：hfetch <num_bytes>, < addr>
参数：num_bytes 为读取数据的字节数
addr 为存储源地址
描述：从 addr 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，该操作会影响 contr 寄存器的值。

hfetcht：

格式：hfetcht <num_bytes>, < addr>
参数：num_bytes 为读取数据的字节数
addr 为存储源地址的寄存器
描述：从 addr 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 contr 寄存器的值。

store：

格式：store <num_bytes>,<addr>
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，该操作会影响 contw 寄存器的值。

storet：

格式：storet <num_bytes>,<addr>
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 temp 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 temp 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，该操作会影响 contw 寄存器的值。

istore：

格式：istore <num_bytes>,<reg>
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 存储的地址
备注：istore 的操作为间接寻址操作，该操作会影响 contw 寄存器的值。

istoret:

格式：hstore <num_bytes>,<reg>
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 num_byte 字节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 存储的地址
备注：istoret 的操作为间接寻址操作，该操作会影响 contw 寄存器的值。

hstore:

格式：hstore <num_bytes>,<addr>

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，该操作会影响 contw 寄存器的值。

hstoret:

格式：hstoret <num_bytes>,<addr>
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 temp 中读取指定长度 num_byte 字节数据写入指定地址 addr
示例：hstoret 2,0x8848 //从 temp 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，该操作会影响 contw 寄存器的值。

force:

格式：force <immediate>,<regs>
参数：Immediate 为立即数，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：none

iforce：

格式：iforce <regw>
参数：regw 为目标寄存器
描述：把 pdata 的数据赋值给 regw
示例：iforce temp //将 pdata 的数据赋值给寄存器 temp。
备注：none

arg：

格式：arg <immediate>,<reg>
参数：immediate 立即数
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 <immediate>
参数：Immediate 为立即数
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 <regr>,<regw>
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

icopy :

格式：icopy <regw>
参数：regw 目标寄存器
描述：通过将 pdata 值复制到 regw 中去
示例：icopy rega //将 pdata 的值赋值给 rega
备注: none

deposit:

格式：deposit <reg>
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: none

disable：

格式：disable <flag>
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 <flag>
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 <addr>,<flag>
参数：addr 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：区别于 ncall

ncall

格式：ncall <addr>,<flag>
参数：addr 为目标地址
flag 为判据,缺省时即不跳转
描述：当 flag = 0 时，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：ncall clean_mem,zero //当 zero = 0 时，跳到 clean_mem 函数中
备注：none

rtn：

格式：rtn <flag>
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：

```
eeprom_load_reconn_info:
    .....
    rtn
```

备注：区别于 nrtn

nrtn：

格式：nrtn <flag>
参数：flag 为判断依据
描述：flag 为 0 则返回，若 flag 为 1 则不返回。
备注：区别于 rtn

rtneq：

格式：rtneq <immediate>
参数：immediate 为指定数值。
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
rtneq 7 //判断寄存器 pdata 是否为 7,若为 7 则返回，否则继续执行。
备注：none

rtnne

格式：rtnne <immediate>
参数：immediate 为指定数值。
描述：若寄存器 pdata 的数值与 immediate 值不相等则返回，否则继续执行。
示例：fetch 1,mem_sp_calc
rtnne 7 //判断寄存器 pdata 是否为 7,若不为 7 则返回，否则继续执行。
备注：none

rtnbit1：

格式：rtnbit1 <immediate>
参数：immediate 为指定寄存器 pdata 的位。
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 <immediate>
参数：immediate 为指定寄存器 pdata 的位。
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

rtnmark1:

格式：rtnmark1 <immediate>

参数：immediate 为指定寄存器 mark 的位。
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
rtnmark1 7 //判断寄存器 mark 的第七位是否为 1，若为 1 则返回，否则继续执行。
备注：none

rtnmark0:

格式：rtnmark0 <immediate>
参数：immediate 为指定寄存器 mark 的位。
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
rtnmark0 7 //判断寄存器 mark 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 <addr>，<flag>
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>
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 <addr>，<flag>
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
备注：none

bbit1：

格式：bbit1 <immediate> ,< addr>
参数：immediate 为指定 pdata 位的值
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 <immediate> ,< addr>
参数：immediate 为指定 pdata 位的值
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

bmark0：

格式：bmark0 <immediate> ,< addr>
参数：immediate 为指定 mark 位的值
addr 为跳转地址
描述：如果 mark 的第 immediate 位为 0，则将跳转到 addr 继续执行
示例：bmark0 3,init_uuid_kb //如果 mark 的第 3 位为 0，则跳转到 init_uuid_kb
备注：none

bmark1：

格式：bmark1 <immediate> ,< addr>
参数：immediate 为指定 mark 位的值
addr 为跳转地址
描述：如果 mark 的第 immediate 位为 1，则将跳转到 addr 继续执行
示例：bmark1 3,init_uuid_kb //如果 mark 的第 3 位为 1，则跳转到 init_uuid_kb
备注：none

beq:

格式：beq <immediate>,<addr>
参数：immediate 为立即数
addr 为跳转地址
描述：若 immediate 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 <immediate> ,<addr>
参数：immediate 为立即数
addr 为跳转地址
描述：若 immediate 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 <addr>
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 <immediate>,<addr>
参数：Immediate 立即数
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 <source>,<dest>,<immediate>
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
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

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的，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：

<判断指令> <条件>；

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 相 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。