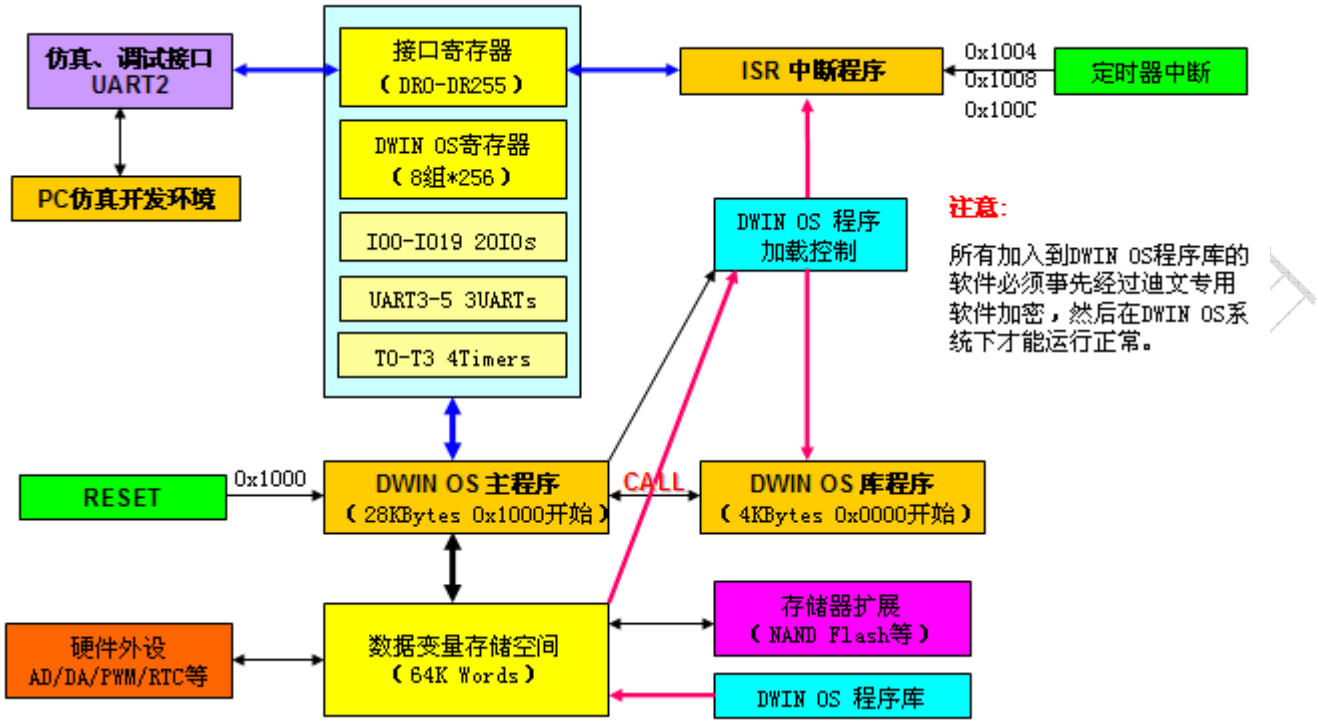
目 录

1 DWIN OS 平台架构



(1) DWIN OS 代码空间定义

代码地址	定 义	说 明
0x0000-0x0FFF	L2_Cache	程序动态加载调用使用空间，4KB。
0x1000	RESET	复位后程序运行首地址，放一条 GOTO 指令跳转到主程序。
0x1004	T0 _INT	T0 INT 程序入口地址，使用 GOTO 指令跳转到 T0 中断服务程序。
0x1008	T1 _INT	T1 INT 程序入口地址，使用 GOTO 指令跳转到 T1 中断服务程序。
0x100C	T2 _INT	T2 INT 程序入口地址，使用 GOTO 指令跳转到 T2 中断服务程序。
0x1020-0x107F	Reserved	保留
0x1080-0x7FFF	Main Code	主程序代码空间

(2) 子程序嵌套调用（含中断程序）最大为 127 级。

(3) 典型程序架构

```
        ORG     1000H
    GOTO  MAIN              ;代码的第一条指令必须是 GOTO 。
    GOTO    T0INT           ;中断产生时，跳转到 T0 中断处理程序，必须使用 GOTO，不能用 CALL。
    NOP                     ;T1 中断未使用
    GOTO    T2INT           ;中断产生时，跳转到 T2 中断服务程序
        ORG     1080H
    MAIN:   NOP             ;主程序开始
            GOTO    MAIN

    T0INT:  NOP             ;T0 中断处理程序
            RETI            ;必须使用 RETI 结束，不能使用 RET。

    T2OINT: NOP             ;T1 中断处理程序
            RETI
```

如果不使用中断（关闭中断），那么 0x1004-0x107F 的代码空间可以任意使用。
如果主程序需要运行断点仿真，一定要关闭中断，不然由于仿真状态下定时器仍旧在正常运行，开启中断将导致主程序无法断点仿真。

2 DWIN OS 调试接口（UART2）

系统调试串口UART2模式固定为8N1，波特率可以设置，数据帧由5个数据块组成：

数据块	1	2	3	4	5
定　义	帧 头	数据长度	指令	数据	CRC 校验（可选）
数据长度	2	1	1	N	2
说　明	0x5AA5	包括指令、数据、校验。	0x80/0x81/0x82/0x83		
举例（无校验）	5A A5	04	83	00 10 04	
举例（带校验）	5A A5	06	83	00 10 04	25 A3

UART2 调试接口指令说明如下：

指令	数　据	说　明
0x80	下发： 寄存器页面（0x00-0x08）+寄存器地址(0x00-0xFF)+写入的数据	指定地址开始写数据串到寄存器。
	应答： 0x4F 0x4B 。	写指令应答。
0x81	下发： 寄存器页面（0x00-0x08）+寄存器地址(0x00-0xFF)+读取数据字节长度(0x01-0xFB)	从指定寄存器开始读数据。
	应答： 寄存器页面（0x00-0x08）+寄存器地址(0x00-0xFF)+数据长度+数据	数据应答。
0x82	下发： 变量空间首地址(0x0000-0xFFFF)+写入的数据	指定地址开始写数据串(字数据)到变量空间。 系统保留的空间不要写。
	应答： 0x4F 0x4B 。	写指令应答。
0x83	下发： 变量空间首地址(0x0000-0xFFFF)+读取数据字长度(0x01-0x7D)	从变量空间指定地址开始读指定长度字数据。
	应答： 变量空间首地址+变量数据字长度+读取的变量数据	数据应答。

寄存器页面定义如下：

寄存器页面 ID	定　义	说　明
0x00-0x07	数据寄存器	每组 256 个，R0-R255
0x08	接口寄存器	DR0-DR255，详见 3.4 接口寄存器定义 说明。

3 存储器空间

3.1 用户数据库

用户数据库包括两部分：

（1）T5L 片内的 Flash，通过系统变量接口访问，所有基于 T5 的 DWIN OS 都支持。

（2）位于片外 Flash 上的大容量数据库或资料存储器，通过系统变量接口访问，空间大小取决于硬件平台。

3.2 数据变量空间

数据变量空间是一个最大 128Kbytes 的双口 RAM，两个 CPU 核通过数据变量空间交换数据，分区定义列表如下：

变量地址区间	区间大小(Kwords)	定 义	说 明
0x0000-0x03FF	1.0	系统变量接口	硬件、存储器访问控制、数据交换。具体定义和硬件平台有关。
0x0400-0x07FF	1.0	系统保留	用户不要使用。
0x0800-0x0BFF	1.0	系统保留	用户不要使用。
0x0C00-0x0FFF	1.0	系统保留	用户不要使用。
0x1000-0xFFFF	60	用户变量区	用户变量、存储器读写缓冲区等，用户自行规划。

在所有基于 T5L CPU 的 DWIN OS 平台上，系统变量接口的前 16 个字定义是统一的，如下表：

地址	定义	长度	说 明
0x00	保留	4	
0x04	System_Reset	2	0x55AA 5AA5=Reset T5L ；
0x06	OS_Update_CMD	2	D3：0x5A 启动一次更新 DWIN OS 程序操作（写到片内 Flash），CPU 操作完清零。 D2：固定为 0x10。代码必须从 0x1000 开始。 D1:0：存储升级代码的数据变量空间首地址，0x1000-0x0C7E，必须是偶数。
0x08	NOR_ Flash_RW_CMD	4	D7：操作模式 0x5A=读 0xA5=写，CPU 操作完清零。 D6:4：片内 Nor Flash 数据库首地址，必须是偶数，0x000000-0x02:FFFE，192KWords。 D3:2：数据变量空间首地址，必须是偶数。 D1:0：读写字长度，必须是偶数。
0x0C	UART2_Set	2	D3=0x5A 启动 UART2 串口模式设置，只用于 GUI CPU 在 Reset 之后来设置 UART2 的模式，OS 本身不能操作。 D2=串口模式，0x00=8N1。 D1:D0=波特率设置值，波特率设置值=3225600/设置的波特率。
0x0E	保留	1	
0x0F	Ver	1	应用软件版本。D1 表示 CPU0 软件版本，D0 表示 CPU1 软件版本。

由于变量存储器是两个 CPU 核共用的存储器，过于频繁的不停读写变量存储器会严重影响 CPU 的处理效率，推荐使用 1mS 定时器中断来定时查询变量存储器的数据更新。

3.3 寄存器

基于 T5L 的 DWIN OS 一共有 2048 个寄存器，分成 8 页来访问，每页 256 个寄存器，对应 R0-R255。

3.4 接口寄存器

基于 T5L 的 DWIN OS 有一个接口寄存器页，一共有 256 个接口寄存器，用于对硬件资源的快速访问接口。

DR#	长度	R/W	定 义	说 明
0	1	R/W	REG_Page_Sel	OS 的 8 个寄存器页切换，DR0=0x00-0x07。
1	1	R/W	SYS_STATUS	系统状态寄存器，按位定义： .7 CY 进位标记。 .6 DGUS 屏变量自动上传功能控制 1=关闭 0=开启。
2	14	--	系统保留	禁止访问。
16	1	R	UART3_TTL_Status	串口接收帧超时定时器状态： 0x00=接收超时定时器溢出 其它=未溢出。 必须先用 RDXLEN 指令读取接收长度，长度不为 0 再检查超时定时器状态。
17	1	R	UART4_TTL_Status	
18	1	R	UART5_TTL_Status	
19	*1*	*R*	*UART6_TTL_Status*	
20	*1*	*R*	*UART7_TTL_Status*	
21	1	--	保留	
22	1	R	UART3_TX_LEN	UART3 发送缓冲区使用深度（Bytes），缓冲区大小为 256Bytes，用户只读。
23	1	R	UART4_TX_LEN	UART4 发送缓冲区使用深度（Bytes），缓冲区大小为 256Bytes，用户只读。
24	1	R	UART5_TX_LEN	UART5 发送缓冲区使用深度（Bytes），缓冲区大小为 256Bytes，用户只读。
25	*1*	*R*	*UART6_TX_LEN*	*UART6 发送缓冲区使用深度（Bytes），缓冲区大小为 256Bytes，用户只读。*
26	*1*	*R*	*UART7_TX_LEN*	*UART7 发送缓冲区使用深度（Bytes），缓冲区大小为 256Bytes，用户只读。*
27	1	--	保留	
28	1	R/W	UART3_TTL_SET	UART3 接收帧超时定时器时间，单位 0.5mS，0x01-0xff，上电设置为 0x0A。
29	1	R/W	UART4_TTL_SET	UART4 接收帧超时定时器时间，单位 0.5mS，0x01-0xff，上电设置为 0x0A。
30	1	R/W	UART5_TTL_SET	UART5 接收帧超时定时器时间，单位 0.5mS，0x01-0xff，上电设置为 0x0A。
31	*1*	*R/W*	*UART6_TTL_SET*	*UART6 接收帧超时定时器时间，单位 0.5mS，0x01-0xff，上电设置为 0x0A。*
32	*1*	*R/W*	*UART7_TTL_SET*	*UART7 接收帧超时定时器时间，单位 0.5mS，0x01-0xff，上电设置为 0x0A。*
33	1	--	保留	
34	1	R/W	T0	8bit 用户定时器 0，++计数，基准 10uS。
35	2	R/W	T1	16bit 用户定时器 1，++计数，基准 10uS。
37	2	R/W	T2	16bit 用户定时器 2，++计数，基准由用户用 CONFIG 指令设定。
39	2	R/W	T3	16bit 用户定时器 3，++计数，基准由用户用 CONFIG 指令设定。
41	*1*	*R/W*	*CNT0_Sel*	*相应位 置 1 选择对应 I/O 进行跳变计数，对应 IO7-IO0。*
42	*1*	*R/W*	*CNT1_Sel*	*相应位 置 1 选择对应 I/O 进行跳变计数，对应 IO7-IO0。*
43	*1*	*R/W*	*CNT2_Sel*	*相应位 置 1 选择对应 I/O 进行跳变计数，对应 IO15-IO8。*
44	*1*	*R/W*	*CNT3_Sel*	*相应位 置 1 选择对应 I/O 进行跳变计数，对应 IO15-IO8。*
45	1	R/W	Int_Reg	中断控制寄存器： .7=中断总开关 1=使能（是否开启取决于单独中断控制位） 0=禁止。 .6=Timer INT0 Enable 1=中断定时器 0 中断开启 0=中断定时器 0 中断关闭。 .5=Timer INT1 Enable 1=中断定时器 1 中断开启 0=中断定时器 1 中断关闭。 .4=Timer INT2 Enable 1=中断定时器 2 中断开启 0=中断定时器 2 中断关闭。
46	1	R/W	Timer INT0 Set	8bit 定时器中断 0 设置值，中断时间=Timer_INT0_Set*10uS，0x00=256。
47	1	R/W	Timer INT1 Set	8bit 定时器中断 1 设置值，中断时间=Timer_INT1_Set*10uS，0x00=256。
48	2	R/W	Timer INT2 Set	16bit 定时器中断 2 设置值，中断时间=（Timer_INT2_Set+1）*10uS。
50	*10*	*R/W*	*Polling_Out0_Set*	*第 1 路 IO0-IO15 定时扫描输出配置，每个配置 10 个字节：* *D9（DR50）：0x5A=扫描输出使用，其它为不使用。* *D8：输出数据的寄存器页面，0x00-0x07。* *D7：输出数据的起始地址，0x00-0xFF。* *D6：输出数据的字长度，0x01-0x80，每个数据 2 个 Byte 对应 IO15-IO0。* *D5-D4：IO15-IO0 输出通道选择，需要输出的通道，相应 bit 设置为 1。* *D3-D2：单步输出间隔 T，单位为（T+1）*10uS。* *D1-D0：输出周期计数设定，每完成 1 个周期输出后减 1，减到 0 后输出为 0。*
60	*10*	*R/W*	*Polling_Out1_Set*	*第 2 路 IO0-IO15 定时扫描输出配置。*
70	9	--	保留	
80	*6*	*R/W*	*IO6 触发时间*	*D5=0x5A 表示捕捉到一次 IO6 下跳沿触发。* *D4:D3=触发时 IO15-IO0 的状态。* *D2:D0=捕捉的系统定时器时间，0x000000-0x00FFFF 循环，单位是 1/41.75uS。*
86	*6*	*R/W*	*IO7 触发时间*	*D5=0x5A 表示捕捉到一次 IO7 下跳沿触发。* *D4:D3=触发时 IO15-IO0 的状态。* *D2:D0=捕捉的系统定时器时间，0x000000-0x00FFFF 循环，单位是 1/41.75uS。*
92	37	--	保留	
129	3	R/W	IO_Status	IO19-IO0 的实时状态。
132	*2*	*R/W*	*CNT0*	*CNT0 跳变计数值，计到 0xFFFF 后复位到 0x0000。*
134	*2*	*R/W*	*CNT1*	*CNT1 跳变计数值，计到 0xFFFF 后复位到 0x0000。*

136	*2*	*R/W*	*CNT2*	*CNT2 跳变计数值，计到 0xFFFF 后复位到 0x0000。*
138	*2*	*R/W*	*CNT3*	*CNT3 跳变计数值，计到 0xFFFF 后复位到 0x0000。*
140	116	--	保留	

4 DWIN OS 汇编指令集

- R#表示 DWIN OS 当前寄存器页内，任意 256 个寄存器之任意一个或一组，R0-R255；
- DR#表示 256 个接口寄存器之任意一个或一组，DR0-DR255；
- < >表示立即数，汇编代码中，100，0x64，64H，064H 都是表示 10 进制数据 100。
- 伪指令：ORG DB DW 。
- 使用 ；做为注释符号。
- DWIN OS 可以访问的变量和数据类型说明如下表：

变量类型	标 记	访问类型	空间大小	说 明
DWIN OS 寄存器	R0-R255	Byte	2048 Bytes	分成 8 页，由 DR0 接口寄存器控制分页。
接口寄存器	DR0-DR255	Byte	256 Bytes	
数据变量空间	XRAM	Word	64K Words	地址范围：0x0000-0xFFFF。
用户数据库	LIB	Word	取决于硬件	

- 在 T5L CPU 运行速度为 200MHz，1 条 DWIN OS 指令的平均执行时间大约 125nS（8MIPS）。

4.1 数据交换指令

指令功能	操作码	操作数	说　　明
数据变量和寄存器数据交换	MOVXR	R#,<MOD>,<NUM>	R#：寄存器或寄存器组。 <MOD>：0=寄存器到变量　1=变量到寄存器。 <NUM>：交换的数据字（Word）长度，0x00-0x80； 当<NUM>为 0x00 时，数据长度由 R9 决定。 数据变量指针由 R0:R1 寄存器定义。 MOVXR R20,0,2
装载 N 个 8bit 立即数到寄存器组	LDBR	R#,<DATA>,<NUM>	R#：寄存器或寄存器组。 <DATA>：要装载的数据。 <NUM>：要装载的寄存器个数，0x00 表示 256 个。 LDBR　R8,0x82,3
装载 1 个 16bit 立即数到寄存器组	LDWR	R#,<DATA>	R#：寄存器组。 <DATA>：要装载的数据。 LDWR　R8,1000 LDWR　R8,-300
加载代码空间地址	LDADR	<Address>	把<Address>加载到 R5:R6:R7 LDADR TAB LDADR 0x123456
程序空间查表（程序空间数据到寄存器）	MOVC	R#,<NUM>	R#：寄存器或寄存器组。 <NUM>：查表返回的字节数据长度。 表地址指针由 R5:R6:R7 寄存器定义。 MOVC R20,10 注意，0x1000 之后的代码不能读取 0x1000 之前的代码内容。
寄存器和寄存器数据交换	MOV	R#S,R#T,<NUM>	R#S：源寄存器或寄存器组。 R#T：目标寄存器或寄存器组。 <NUM>：交换的字节数据长度，0x00 表示长度由 R9 寄存器定义。 MOV R8,R20,3
寄存器到接口寄存器	MOVRD	R#,DR#,<NUM>	R#：寄存器或寄存器组； DR#：接口寄存器或寄存器组； <NUM>：交换的字节数据长度，0x00 表示长度由 R9 寄存器定义。 MOVRD R10,3,2
接口寄存器到寄存器	MOVDR	DR#,R#,<NUM>	R#：寄存器或寄存器组； DR#：接口寄存器或寄存器组； <NUM>：交换的字节数据长度，0x00 表示长度由 R9 寄存器定义。 MOVDR 3,R10,2
数据变量间交换数据	MOVXX	<NUM>	<NUM>：交换（字，Word）数据长度。 <NUM>为 0 表示长度由 R8:R9 寄存器定义。 源变量地址由 R0:R1 寄存器定义。 目标变量地址由 R2:R3　寄存器定义。 当源地址和目标地址间距小于移动数据长度时，移动数据长度不能超过 32。 MOVXX 100
存储器变址寻址	MOVA	无或 0x00	寄存器数据交换，按照寄存器分页访问： R2 规定了源寄存器（组）地址，R3 规定了目标寄存器（组）地址； R9 规定了交换的数据长度，字节数。 MOVA　或　MOVA 0x00
		0x01	寄存器数据交换，把所有寄存器看成 1 个 2KB 数据区来访问： R0:R1 规定了源寄存器地址，0x0000-0x7FF； R2:R3 规定了目标寄存器地址，0x0000-0x7FF； R9 规定了交换的数据长度，字节数，0x00-0xFF，0x00 表示 256。 地址高字节=源或目标寄存器 DR0　低字节=源或目标寄存器地址　。 MOVA 0x01
寄存器入栈	PUSH	R#,<NUM>	把 R#开始的<NUM>个寄存器数据保存到数据堆栈。 PUSH R8,4
寄存器出栈	POP	R#,<NUM>	从数据堆栈取出数据到 R#开始的<NUM>个寄存器。 POP R8,4

数据堆栈深度为 256 个字节。

4.2 数学运算指令

指令功能	操作码	操作数	说　　明
32bit 整形数加法	ADD	R#A,R#B,R#C	C=A+B,A、B 为 32bit 整数；C 为 64bit 整数。 ADD R10,R20,R30
32bit 整形数减法	SUB	R#A,R#B,R#C	C=A-B,A、B 为 32bit 整数；C 为 64bit 整数。 SUB R10,R20,R30
64bit 长整数 MAC	MAC	R#A,R#B,R#C	C=(A*B+C)，A、B 是 32bit 整数，C 是 64Bit 整数。 MAC R10,R20,R30
64bit 整形数除法	DIV	R#A,R#B,<MOD>	A/B，商是 A，余数是 B，A 和 B 都是 64bit 寄存器。 <MOD>：0=商不进行四舍五入，1=商进行四舍五入。 DIV R10,R20,1
变量扩展成 32bit	EXP	R#S,R#T,<MOD>	把 R#S 指向的数据转成 32bit 整数保存到 R#T。 R#S：源寄存器或寄存器数。 R#T：32bit 目标寄存器。 <MOD>：R#S 数据类型，0=8bit 无符号 1=8bit 带符号 2=16bit 无符号数 3=16bit 整数。 EXP R10,R20,2
32bit 无符号 MAC	SMAC	R#A,R#B,R#C	C=A*B+C。 A、 B 是 16bit 无符号数，C 是 32bit 无符号数。 SMAC R10,R20,R30
寄存器自增量	INC	R#，<MOD>,<NUM>	R#=R#＋NUM，无符号数自增计算，<NUM>为 0x00-0xFF。 <MOD>：R#数据类型，0=8bit 1=16bit。 INC R10,1,5
寄存器自减量	DEC	R#,<MOD>,<NUM>	R#=R#-NUM，无符号数自减计算，<NUM>为 0x00-0xFF。 <MOD>：R#数据类型，0=8bit 1=16bit。 DEC R10,0,1
平方根计算	SQRT	R#A,R#B	计算一个 64 位无符号数 R#A 的平方根并保存到 R#B 中。 R#A：保存了 8 Bytes 无符号数； R#B：保存了 4 Bytes 无符号数结果。 SQRT R80,R90
浮点数和定点整数转换	FINT	R#F,R#I,<MOD>	实现 1 个浮点数和 1 个 64bit 的定点整数转换。 R#F：保存浮点数的寄存器，32bit 或 64bit； R#I：保存定点整数的寄存器，64bit； <MOD>： .7 表示浮点数格式： 0=32bit 单精度 1=64bit 双精度。 .6 转换类型 0=浮点数转换成定点整数 1=定点整数转换成浮点数。 .5 未定义，写 0 .4-.0 定点整数的小数位数，0x00-0x1F，最多 31 位小数。

4.3 逻辑运算指令

指令功能	操作码	操作数	说　　明
与逻辑运算	AND	R#A,R#B,<NUM>	A=A AND B，序列与逻辑运算。 <NUM>：R#A、R#B 变量字节数目。 AND R10,R20,1
或逻辑运算	OR	R#A,R#B,<NUM>	A=A OR B，序列或逻辑运算。 <NUM>：R#A、R#B 变量字节数目。 OR R10,R20,1
异或逻辑运算	XOR	R#A,R#B,<NUM>	A=A XOR B，序列异或逻辑运算。 <NUM>：R#A、R#B 变量字节数目。 XOR R10,R20,1
左环移	SHL	R#,<NUM>,<BIT_NUM>	把 R#指向的<NUM>个寄存器向左环移<BIT_NUM>位。 SHL R10,2,1
右环移	SHR	R#,<NUM>,<BIT_NUM>	把 R#指向的<NUM>个寄存器向右环移<BIT_NUM>位。 SHR R10,2,1

4.4 数据处理指令

指令功能	操作码	操作数	说　　明
序列比较	TESTS	R#A,R#B,<NUM>	依次比较 A、B 两个寄存器序列的值： 值不同时，返回 A 序列此时的地址到 R0 寄存器； 如果 A、B 相同则返回 0x00 到 R0 寄存器。 R#A：A 序列寄存器。 R#B：B 序列寄存器。 <NUM>：最大比较数据字节长度。 TESTS R10,R20,16
解整数线性方程	ROOTLE		由 16bit 整数（X0,Y0）、（X1,Y1）两点确定的直线上的 X，求解对应的 Y 值。 输入：X=R16 X0=R20 Y0=R22 X1=R24 Y1=R26。 输出：Y=R18。 ROOTLE
ANSI　CRC-16 计算	CRCA	R#S,R#T,R#N	对序列数据计算 ANSI CRC-16(X16+X15+X2+1）。 R#S：输入的寄存器组； R#T：输出 CRC 结果，16bit，LSB 模式保存。 R#N：保存计算 CRC 数据字节长度寄存器，8bit。 CRCA R10,R80,R9
CCITT CRC-16 计算	CRCC	R#S,R#T,R#N	对序列数据计算 CCITT CRC-16(X16+X12+X5+1）。 R#S：输入的寄存器组； R#T：输出 CRC 结果，16bit，MSB 模式保存。 R#N：保存计算 CRC 数据字节长度寄存器，8bit。 CRCC R10,R80,R9
HEX 转 ASCII 字符串	HEXASC	R#S,R#T,<MOD>	R#S：需要转换的 32bit 整数； R#T：转换后的 ASCII 字符串寄存器组； <MOD>：转换模式控制，高 4bit 为整数位长度，低 4bit 为小数位数目。 转换的 ASCII 串带符号，右对齐，空位用 0x20 填充。 对于数据 0x12345678， <MOD>=0x62　转换结果为+054198.96 <MOD>=0xF2　转换结果为　　+3054198.96 HEXASC R20,R30,0x62
ASCII 字符串转 HEX	ASCHEX	R#S,R#T,<LEN>	把 ASCII 字符串转换成 64bit 整数。 R#S：输入 ASCII 字符串寄存器首地址； R#T：输出 HEX 数据，64bit 寄存器； <LEN>：ASCII 字符串数据长度，包括符号位和小数点，0x01-0x15。 ASCHEX R10,R80,0x05
数据处理	MATH	<MOD>,R#P,R#N	对数据存储区的数据进行处理，数据是 16bit 无符号数。 R#P：R#P 开始的 7 个寄存器数据 D0-D7。 D0 处理结果保存的寄存器首地址。 D1:D2 下一个（未来）数据在数据缓冲区的首地址(**相对地址**，数据处理时会从当前位置向前读历史数据)。 D3:D4 数据缓冲区在数据存储区的首地址。 D5:D6 数据缓冲区的字（Word）长度，必须不小于 R#N 的值。 R#N：本次数据处理点数（每点 1-N 字），0x00-0xFF，0x00 表示 256。 <MOD>=0x00 计算平均值，结果为 32bit 无符号数，单位为 1/65536。 <MOD>=0x01 计算最大值，结果为 16bit 无符号数。 <MOD>=0x02 计算最小值，结果为 16bit 无符号数。 <MOD>=0x03 计算均方根值（RMS），结果为 32bit 无符号数，单位为 1/65536。 <MOD>=0x04 按照 y=k*x+b 进行最小二乘法参数估计。 数据存储格式是（x0,y0）....（xn,yn），返回格式是（k,b）。 k,b 返回结果为 32bit 整数，单位为 1/65536。 <MOD>=0x05 不支持。 <MOD>=0x06 计算标准差值（RMSE），结果为 32bit 无符号数，单位为 1/65536。 MATH 0,R0,R10

4.5 进程控制指令

指令功能	操作码	操作数	说　　明
空操作	NOP		不执行任何操作。 NOP
位测试、跳转	JB	R#,<Bit>,<TAB>	测试 R#指向的 16bit 寄存器的第<Bit>位，1 跳转，0 继续执行下一条代码，跳转范围+/-127 条指令。 R#：位测试的寄存器，16bit。 <Bit>：位测试位置，0x00-0x0F，MSB 方式。 <TAB>：跳转位置。 JB R10,15,TEST1 NOP TEST1:ADD R8,R12,R16
变量比较、不相等跳转	CJNE	R#A,R#B,<TAB>	比较 A、B 两个 8bit 寄存器的内容，相等则执行下一条指令，不等则跳转，跳转范围+/-127 条指令。 TEST1:NOP INC R10,0,1 CJNE R10,R11,TEST1
16bit 整形数比较，小于跳转	JS	R#A,R#B,<TAB>	比较 A、B 两个 16bit 整数的大小，A>=B 则执行下一条指令，A<B 则跳转，跳转范围+/-127 条指令。 JS R10,R12,TEST1 NOP TEST1:NOP
16bit 无符号数比较，小于跳转	JU	R#A,R#B,<TAB>	比较 A、B 两个 16bit 无符号数的大小，A>=B 则执行下一条指令，A<B 则跳转，跳转范围+/-127 条指令。 JU R10,R12,TEST1 NOP TEST1:NOP
变量和立即数比较、不相等跳转	IJNE	R#,<INST>,<TAB>	比较 8bit 寄存器和立即数<INST>的内容，相等则执行下一条指令，不等则跳转，跳转范围+/-127 条指令。 IJNE R10,100,TEST1 NOP TEST1:NOP
自减大于 0 跳转	DJNZ	R#,<NUM>,<TAB>	R#是 16bit 无符号数，每次计算 R#=R#-<NUM>，如果 R#大于 0 则跳转，反之执行下一条指令，跳转范围+/-127 条指令。 TEST1:NOP DJNZ R10,1,TEST1
子程序调用返回	RET		CALL 调用指令返回。 RET
中断程序返回	RETI		中断服务程序返回返回。 RETI
子程序调用	CALL	<PC>	调用子程序，最多支持 32 级程序嵌套。 CALL TEST
直接跳转	GOTO	<PC>	程序跳转。 如果<PC>=0xFFFF 表示以<R5:R6:R7>位置为基准，R1:R0 做为相对 PC 指针来跳转。 GOTO TEST1 NOP TEST1:NOP
程序结束	END		DWIN OS 程序运行结束指令，执行本指令后 PC 指针复位到 0x1000 重新运行。相当于软件复位。 END

注意：
中断程序使用必须使用 GOTO 、RETI 指令。
子程序的调用必须成对的使用 CALL 和 RET 指令，使用 GOTO 和 RET 指令调用程序将导致堆栈溢出异常。

4.6 外设操作指令

指令功能	操作码	操作数	说　　明
串口配置	COMSET	<MODE/R#>,<BS>	设置串口模式： <MODE>：高 4bit 选择要配置的串口，3=UART3 … 5=UART5 低 4bit 选择模式 0x*0=N81，0x*1=E81，0x*2=O81，0x*3=N82 模式。 <BS>:波特率设置值，2Bytes。 对于 UART3， 设置值=6451200/设置的波特率，设置值范围=1-1023。 对于 UART4-UART5， 设置值=25804800/设置的波特率，设置值范围 1-65535。 每次设置会清空对应 UART 的收发缓冲区。 如果<BS>=0x0000，那么<MODE>将是寄存器指针，指针指向的 3 个寄存器顺序对应<MODE> 、<BS>的值 。 COMSET 0x30,136
串口发送	COMTXD	<COM>,R#S,R#N	把数据发送到指定的串口。 <COM>：选择串口，**0-2 不支持，**3=UART3… 5=UART5 R#S：要发送的数据寄存器组。 R#N：要发送的字节数寄存器，8bit，寄存器数据 0x00 表示发送 256 字节数据。 COMTXD 3,R10,R9
检查 COM_Rx_FIFO	RDXLEN	<COM>,R#	返回 COM 接收缓冲区（FIFO）接收数据字节长度（0-255）到 R#寄存器，0x00 表示没有数据。 <COM>：选择串口，**0-2 不支持，**3=UART3… 5=UART5 RDXLEN 3,R10
读取 COM_Rx_FIFO	RDXDAT	<COM>,R#A,R#B	从 COM 接收缓冲区（FIFO）中读取 R#B 个字节（01-255）到 R#A 寄存器组；读取后 FIFO 长度自动调整。 <COM>：选择串口，**0-2 不支持，**3=UART3… 5=UART5 RDXDAT 3,R11,R10
直接串口发送	COMTXI	<COM>,R#,<NUM>	把 R#指向的<NUM>个寄存器内容发送到 COM。 <COM>：选择串口，**0-2 不支持，**3=UART3… 5=UART5 COMTXI 3,R20,16
硬件配置	CONFIG	<TYPE>,<D1/R#>,<D0>	<TYPE>：硬件类型选择，仅低 7bit 有效。 TYPE.7=0 表示 D1、D0 是立即数。 TYPE.7=1 表示 D1 是一个寄存器指针，指针指向的 2 个寄存器顺序对应<D1><D0>的值。 TYPE.6-TYPE.0 选择硬件类型， 0x00：配置 I/O 口模式。 D1 选择 IO 口，0x00-0x02 对应 P0-P2，其中 P1.7-P1.0 对应 IO7-IO0 P2.7-P2.0 对应 IO15-IO8 P3.3-P3.0 对应 IO19-IO16 D0 是相应的配置值。 D0.X=1 为输出（Push-Pull） D0.X=0 为输入（Open Drain）。 输出状态下也可以读入 IO 口状态。 0x01：配置定时器。 D1 选择定时器，0x02-0x03 对应 T2、T3。 D0 设置定时器基准，单位为 1ms，0x00 表示 256。 0x02：加密的 4KB OS LIB 代码加载，D1:D0 是存储代码变量空间地址，必须是偶数；加载到 0x0000 开始的代码空间，用 CALL 0x0000 调用。 0x03：加密的 4KB OS LIB 代码加载，D1:D0 是存储代码的变量空间地址，必须是偶数。加载到 0x1000 开始的代码空间。 0x04：未加密的 512 字节 OS 代码加载（程序调用或进程管理）；**D1:00** 是存储代码的变量空间字地址，**D0:00** 是代码空间字节地址。 加密代码须使用迪文专用工具预先加密，4KB 加载时间 200uS 左右。 CONFIG 0,0,0x0F
IO 操作：输出	OUTPUT	<P#>,<MOD>,R#/<OUT>	向一个指定的 IO 口（8bit 或 1bit）输出。 <P#>：IO 口的编号，0x00-0x02 对应 P1-P3。 <MOD>：输出模式 0x00=8 位输出，输出是立即数<OUT>。 0x01=8 位输出，输出是 R#指定的值。 0x*2=输出 R#.0，并把 R#右环移 1 次，<MOD>高 4bit 是 IO 位置。 0x*3=输出 R#.7，并把 R#左环移 1 次，<MOD>高 4bit 是 IO 位置。 OUTPUT 0,0,0x55 ；P1（IO7-IO0）口输出 01010101

			OUTPUT 1,0x32,R2 ；把 R2.0 输出到 IO11，并把 R2 右环移 1 次
IO 操作：输入	INPUT	<P#>,<MOD>,R#	读取指定 IO 口（8bit 或 1bit）内容到寄存器。 <P#>：IO 口的编号，0x00-0x02 对应 P1-P3。 <MOD>：输入模式 0x00=8 位并行输入。 0x*2=把 R#右环移 1 次，读到 R#.7，<MOD>高 4bit 是 IO 位置。 0x*3=把 R#左环移 1 次，读到 R#.0，<MOD>高 4bit 是 IO 位置。 R#：IO 口数据读取到的寄存器 ID。 INPUT 1,0,R20 ；IO15-IO8 输出 R20 寄存器值 INPUT 1,0x32,R2 ；R2 右环移 1 次，IO11 的状态读入 R2.7。

附录 1 修订记录

日 期	修订内容	软件版本
2019.02.12	首次发布。	V1.0

使用本文档或迪文产品过程中如存在任何疑问，或欲了解更多迪文产品最新信息，请及时与我们联络：
400 免费电话：400 018 9008
企业 QQ 和微信：400 018 9008
企业 mail：dwinhmi@dwin.com.cn

感谢大家一直以来对迪文的支持，您的支持是我们进步的动力！
谢谢大家！