IN1001 GUTTA 内存使用

CREATE: 2008/11/20
UPDATE: 2010/12/03
Version 1.1
GUTTA Ladder Editor Version 1.1
http://www.plcol.com
http://www.visiblecontrol.com

概述

在 GUTTA 平台中，PLC 对内存的访问是按“区域”来进行的。每个区域的内存一般都有特殊的用途。例如 I 区域表示离散量输入、Q 区域表示离散量输出。PLC 内存的区域分布通过变量描述文件定义，详见《UM4002 变量描述文件规范》。这个文件描述了内存区域的数量、大小、每个区域上内存访问的方法以及可以使用的变量形式等。这里介绍内存使用的一般形式。

变量分区

MODBUS 变量分区

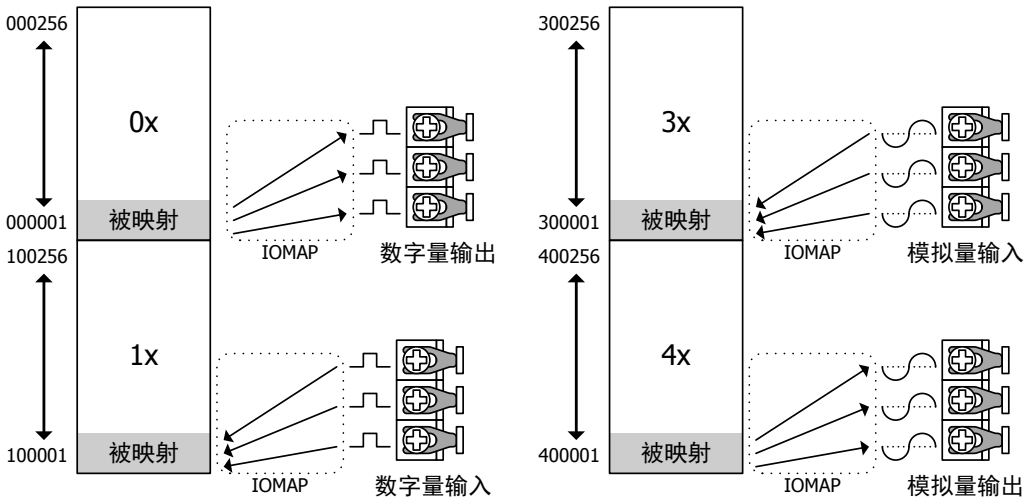
GUTTA PLC 变量区间划分是在 MODBUS 变量区间划分的基础上进行的一种二次划分。PLC 系统在进行 MODBUS 通讯时，依然依据 MODBUS 变量系统对内存的划分进行访问。因此在这里有必要先明确 MODBUS 的变量区间。在标准的 MODBUS PLC 中，一般认为有下面 4 种变量区间。

- 输入线圈（**Input Coil**） - 从地址 10001 开始。
- 保持线圈（**Holding Coil**） - 从地址 00001 开始。
- 输入寄存器（**Input Register**） - 从地址 30001 开始。
- 保持寄存器（**Holding Register**） - 从地址 40001 开始。

其中，输入线圈可以使用 MODBUS 02 通讯指令操作；输出线圈可以使用 MODBUS 01、05、15 通讯指令操作。输入寄存器可以使用 MODBUS 04 通讯指令操作；保持寄存器可以使用 MODBUS 03、06、16 通讯指令操作。通讯指令对输入线圈和输入寄存器只能进行读操作。通讯指令对保持线圈和保持寄存器除了可以进行读操作之外，还能进行写操作。输入线圈和输入寄存器顾名思义，一般被做为离散量输入和模拟量输入的映像。保持线圈和保持寄存器除了被做为离散量输出和模拟量输出的映像之外，还可用来存储 PLC 程序中的中间变量（又称中间线圈或中间寄存器）。在作为输入输出映像的时候，可以通过一个叫做 IO 映射表的部件来配置（又称 IOMAP）。

在 GUTTA 平台中，IO 映射表是固定的，用户不能够任意配置。在需要经常进行 IO 重映射的时候，建议使用符号表来进行 IO 的替换。但这样的替换是静态的，程序重新下载到 PLC 后，配置才生效。因此即使使用了符号表，也不能实现动态的 IO 重映设。

MODBUS 通讯协议中，输入线圈、保持线圈以位为单位操作。以位为单位意味着最小的操作单元是位，并且通讯中的偏移字段代表的也是位偏移。输入寄存器、保持寄存器以字为单位操作（2 字节）。以字为单位操作意味着最小的操作单元是字，并且通讯中的偏移字段代表的也是字偏移。



如上图所示：

保持线圈 000001~000256 的前 3 个线圈 000001~000003 的值被映射到 3 个数字量输出上。保持线圈顾名思义，在程序不对其进行操作时，其值是不会改变的。也就是说，IO 映射表永远只能对保持线圈进行读操作。IO 映射表读取其值后，将线圈的值更新到对应的 PLC 数字量输出端子上（继电器、三极管等形式）。这个更新过程只在 PLC 主程序扫描结束时发生。因此在 PLC 主程序内，保持线圈值的改变不会立即改变对应输出端子的状态。

输入线圈 100001~100256 的前 3 个线圈 100001~100003 的值被映射到 3 个数字量输入上。输入线圈顾名思义，IO 映射表永远只能对输入线圈进行写操作。IO 映射表获得 PLC 数字量输入端子的状态（电压、电流等信号），然后将这个状态的值写入对应的输入线圈。这个更新过程只在 PLC 主程序扫描开始时发生。因此在 PLC 主程序内，输入端子状态的改变不会立即改变输入线圈的值。

保持寄存器 400001~400256 的前 3 个寄存器 400001~400003 的值被映射到 3 个模拟量输出上。保持寄存器顾名思义，在程序不对其进行操作时，其值是不会改变的。也就是说，IO 映射表永远只能对保持寄存器进行读操作。IO 映射表读取其值后，将寄存器的值更新到对应的 PLC 模拟量输出端子上（电压、电流等形式）。这个更新过程只在 PLC 主程序扫描结束时发生。因此在 PLC 主程序内，保持寄存器值的改变不会立即改变对应输出端子的状态。

输入寄存器 300001~300256 的前 3 个寄存器 300001~100003 的值被映射到 3 个模拟量输入上。输入寄存器顾名思义，IO 映射表永远只能对输入寄存器进行写操作。IO 映射表获得 PLC 模拟量输入端子的状态（电压、电流等信号），然后将这个状态对应的值写入对应的输入寄存器。这个更新过程只在 PLC 主程序扫描开始时发生。因此在 PLC 主程序内，输入端子状态的改变不会立即改变输入寄存器的值。

IO 映射表决定 IO 端子状态和 PLC 变量的对应关系。GUTTA PLC 将保持寄存器（4x）区域中没有被映射的部分，作为 PLC 的中间变量使用。

GUTTA PLC 扩展的变量分区

完整的 GUTTA 系统在 MODBUS 提供的 4 类变量空间的基础上，还需要拓展两类变量空间。他们是常数区域和临时区域。由于这两个区域在 MODBUS 定义之外，因此这两个区域的值不能被标准的 MODBUS 通讯协议访问和修改。但是这些区域的值可以由 GUTTA 通讯协议访问和修改。详情可参考文档：《UM4001 GUTTA 通讯协议》。

常数区域用来存放 PLC 程序中使用的常数（指令中的立即数）；临时区域用来存放调用单元（函数调用或中断调用）中使用的临时变量。临时变量也用于传递函数调用的参数。

CPU-EC20 (8051,Compile) 变量分区

MODBUS 地址	槽号	区域标识	区域说明	变量偏移单位	位访问	字节访问	字访问	双字访问	取地址	取值	取指针
输入线圈(1x)	0	I	数字量输入	BYTE	√	√	√	√		√	
保持线圈(0x)	1	Q	数字量输出	BYTE	√	√	√	√		√	
输入寄存器(3x)	2	AI	模拟量输入	BYTE		√	√	√		√	
保持寄存器(4x)	3	AQ	模拟量输出	BYTE		√	√	√		√	
	4	M	普通内存	BYTE	√	√	√	√	√	√	√
	5	T	定时器专用	WORD			√			√	
	6	C	计数器专用	WORD			√			√	
	7	SM	系统内存	BYTE	√	√	√	√	√	√	
	8	J	流程控制专用	BYTE		√				√	
常数区域	9	K	常数区域	BYTE	√	√	√	√		√	
临时区域	10	L	临时区域	BYTE	√	√	√	√	√	√	√

CPU-EC20 (8051,Compile) 变量分区大小

区域	MODBUS 地址开始	MODBUS 地址结束	长度（字节）	范围
I	100001	100064	8	IB0~IB7
Q	000001	000064	8	QB0~QB7
AI	300001	300008	16	AIB0~AIB15
AQ	400001	400008	16	AQB0~AQB15
M	400009	400072	128	MB0~MB127
T	400073	400088	32	T0~T15
C	400089	400096	16	C0~C7
SM	400097	400104	16	SMB0~SMB15
J	400105	400112	16	J0~J15
K			128	--
L			32	LB0~LB31

CPU-EC20 (AVR) 变量分区

MODBUS 地址	槽号	区域标识	区域说明	变量偏移单位	位访问	字节访问	字访问	双字访问	取地址	取值	取指针
输入线圈(1x)	0	I	数字量输入	BYTE	√	√	√	√		√	
保持线圈(0x)	1	Q	数字量输出	BYTE	√	√	√	√		√	
输入寄存器(3x)	2	AI	模拟量输入	BYTE		√	√	√		√	
保持寄存器(4x)	3	AQ	模拟量输出	BYTE		√	√	√		√	
	4	M	普通内存	BYTE	√	√	√	√	√	√	√
	5	T	定时器专用	WORD			√			√	
	6	C	计数器专用	WORD			√			√	
	7	SM	系统内存	BYTE	√	√	√	√	√	√	
	8	J	流程控制专用	BYTE		√				√	
常数区域	9	K	常数区域	BYTE	√	√	√	√		√	
临时区域	10	L	临时区域	BYTE	√	√	√	√	√	√	√

- CPU-EC20 (AVR,Compile) 变量分区同 CPU-EC20 (AVR)。

CPU-EC20 (AVR) 变量分区大小

区域	MODBUS 地址开始	MODBUS 地址结束	长度（字节）	范围
I	100001	100128	16	IB0~IB15
Q	000001	000128	16	QB0~QB15
AI	300001	300008	16	AIB0~AIB15
AQ	400001	400008	16	AQB0~AQB15
M	400009	400264	512	MB0~MB511
T	400265	400296	64	T0~T31
C	400297	400312	32	C0~C15
SM	400313	400328	32	SMB0~SMB31
J	400329	400336	16	J0~J15
K			256	--
L			32	LB0~LB31

- CPU-EC20 (AVR,Compile) 变量分区大小同 CPU-EC20 (AVR)。

CPU-EC20 (Cortex-M3) 变量分区

MODBUS 地址	槽号	区域标识	区域说明	变量偏移单位	位访问	字节访问	字访问	双字访问	取地址	取值	取指针
输入线圈(1x)	0	I	数字量输入	BYTE	√	√	√	√		√	
保持线圈(0x)	1	Q	数字量输出	BYTE	√	√	√	√		√	
输入寄存器(3x)	2	AI	模拟量输入	BYTE		√	√	√		√	
保持寄存器(4x)	3	AQ	模拟量输出	BYTE		√	√	√		√	
	4	M	普通内存	BYTE	√	√	√	√	√	√	√
	5	T	定时器专用	WORD			√			√	
	6	C	计数器专用	WORD			√			√	
	7	SM	系统内存	BYTE	√	√	√	√	√	√	
	8	J	流程控制专用	BYTE		√				√	
常数区域	9	K	常数区域	BYTE	√	√	√	√		√	
临时区域	10	L	临时区域	BYTE	√	√	√	√	√	√	√

- CPU-EC20 (Cortex-M3,Compile) 变量分区同 CPU-EC20 (Cortex-M3)。

CPU-EC20 (Cortex-M3) 变量分区大小

区域	MODBUS 地址开始	MODBUS 地址结束	长度（字节）	范围
I	100001	100128	16	IB0~IB15
Q	000001	000128	16	QB0~QB15
AI	300001	300008	16	AIB0~AIB15
AQ	400001	400008	16	AQB0~AQB15
M	400009	401544	3072	MB0~MB3071
T	401545	401608	128	T0~T63
C	401609	401640	64	C0~C31
SM	401641	401768	256	SMB0~SMB255
J	401769	401784	32	J0~J31
K			256	--
L			32	LB0~LB31

- CPU-EC20 (Cortex-M3,Compile) 变量分区大小同 CPU-EC20 (Cortex-M3)。

EC30-EK51 变量分区

MODBUS 地址	槽号	区域标识	区域说明	变量偏移单位	位访问	字节访问	字访问	双字访问	取地址	取值	取指针
输入线圈(1x)	0	I	数字量输入	BYTE	√	√	√	√		√	
保持线圈(0x)	1	Q	数字量输出	BYTE	√	√	√	√		√	
输入寄存器(3x)	2	AI	模拟量输入	BYTE		√	√	√		√	
保持寄存器(4x)	3	AQ	模拟量输出	BYTE		√	√	√		√	
	4	M	普通内存	BYTE	√	√	√	√	√	√	√
	5	T	定时器专用	WORD			√			√	
	6	C	计数器专用	WORD			√			√	
	7	SM	系统内存	BYTE	√	√	√	√	√	√	
	8	J	流程控制专用	BYTE		√				√	
常数区域	9	K	常数区域	BYTE	√	√	√	√		√	
临时区域	10	L	临时区域	BYTE	√	√	√	√	√	√	√

EC30-EK51 变量分区大小

区域	MODBUS 地址开始	MODBUS 地址结束	长度（字节）	范围
I	100001	100064	8	IB0~IB7
Q	000001	000064	8	QB0~QB7
AI	300001	300012	24	AIB0~AIB23
AQ	400001	400012	24	AQB0~AQB23
M	400013	400140	256	MB0~MB255
T	400141	400172	64	T0~T31
C	400173	400180	16	C0~C7
SM	400181	400188	16	SMB0~SMB15
J	400189	400196	16	J0~J15
K			256	--
L			32	LB0~LB31

EC30-EKSTM32 变量分区

MODBUS 地址	槽	区	区域说明	变量	位	字	字	双	取	取	取

	号	域标识		偏移单位	访问	节访问	访问	字访问	地址	值	指针
输入线圈 (1x)	0	I	数字量输入	BYTE	√	√	√	√		√	
保持线圈 (0x)	1	Q	数字量输出	BYTE	√	√	√	√		√	
输入寄存器 (3x)	2	AI	模拟量输入	BYTE		√	√	√		√	
保持寄存器 (4x)	3	AQ	模拟量输出	BYTE		√	√	√		√	
	4	M	普通内存	BYTE	√	√	√	√	√	√	√
	5	T	定时器专用	WORD			√			√	
	6	C	计数器专用	WORD			√			√	
	7	HC	高速计数器	DWORD				√		√	
	8	SM	系统内存	BYTE	√	√	√	√	√	√	
	9	J	流程控制专用	BYTE		√				√	
常数区域	10	K	常数区域	BYTE	√	√	√	√		√	
临时区域	11	L	临时区域	BYTE	√	√	√	√	√	√	√

EC30-EKSTM32 变量分区大小

区域	MODBUS 地址开始	MODBUS 地址结束	长度（字节）	范围
I	100001	100256	32	IB0~IB31
Q	000001	000256	32	QB0~QB31
AI	300001	300128	256	AIB0~AIB255
AQ	400001	400128	256	AQB0~AQB255
M	400129	401664	3072	MB0~MB3071
T	401665	401792	256	T0~T127
C	401793	401856	128	C0~C63
HC	401857	401864	16	HC0~HC3
SM	401865	401992	256	SMB0~SMB255
J	401993	402000	16	J0~J15
K			512	--
L			32	LB0~LB31

EC30-EKSTM32-XC 变量分区

MODBUS 地址	槽号	区域	区域说明	变量偏移单位	位访	字节	字访	双字	取地	取值	取指

		标识			问	访问	问	访问	址		针
输入线圈 (1x)	0	I	数字量输入	BYTE	√	√	√	√		√	
保持线圈 (0x)	1	Q	数字量输出	BYTE	√	√	√	√		√	
输入寄存器 (3x)	2	AI	模拟量输入	BYTE		√	√	√		√	
保持寄存器 (4x)	3	AQ	模拟量输出	BYTE		√	√	√		√	
	4	M	普通内存	BYTE	√	√	√	√	√	√	√
	5	V	扩展内存	BYTE	√	√	√	√	√	√	√
	6	T	定时器专用	WORD			√			√	
	7	C	计数器专用	WORD			√			√	
	8	HC	高速计数器	DWORD				√		√	
	9	SM	系统内存	BYTE	√	√	√	√	√	√	
	10	J	流程控制专用	BYTE		√				√	
常数区域	11	K	常数区域	BYTE	√	√	√	√		√	
临时区域	12	L	临时区域	BYTE	√	√	√	√	√	√	√

EC30-EKSTM32-XC 变量分区大小

区域	MODBUS 地址开始	MODBUS 地址结束	长度（字节）	范围
I	100001	100256	32	IB0~IB31
Q	000001	000256	32	QB0~QB31
AI	300001	300128	256	AIB0~AIB255
AQ	400001	400128	256	AQB0~AQB255
M	400129	401664	3072	MB0~MB3071
V	401665	405760	8192	VB0~VB8191
T	405761	405888	256	T0~T127
C	405889	405952	128	C0~C63
HC	405953	405960	16	HC0~HC3
SM	405961	406088	256	SMB0~SMB255
J	406089	406096	16	J0~J15
K			512	--
L			32	LB0~LB31

其中可以看出，EC20/EC20 变量分区主要是对保持寄存器（4x）区域进行了细分。除了用于 I/O 映射的区域 AQ、用于中间变量的区域 M、还加入了定时器专用区域 T、计数器专用区域 C、系统内存区域 SM、流程控制区域 J、高数计数区域 HC、扩展内存区域 V。EC20/EC20 变量分区拓展了 MODUBS 分区的变量访问方式。在 MODBUS 中，输入线圈（1x）

和保持线圈（0x）都只能按位方式访问。在 EC20/EC20 中，对应的 I、Q 区域不但可以按位方式访问，还可以按字节、字、双字的方式访问。在 MODBUS 中，输入寄存器（3x）和保持寄存器（4x）都只能按字方式访问。在 EC20/EC20 中，对应的 M、SM 区域不但可以按字方式访问，还可以按位、字节、双字的方式访问。

由上面的表格可以知道，每个 PLC 变量地址都有一个与之对应的 MODBUS 通讯地址。由于 PLC 具体实现时对 MODBUS 区域的划分不一样，以前只能通过查阅表格来计算这个地址。这样不仅繁琐而且容易出错。现在可以直接在软件中查询变量的 MODBUS 地址了，您所需要做的就是在 GUTTA Ladder Editor 软件中调用这个对话框，然后输入一个 PLC 变量名。

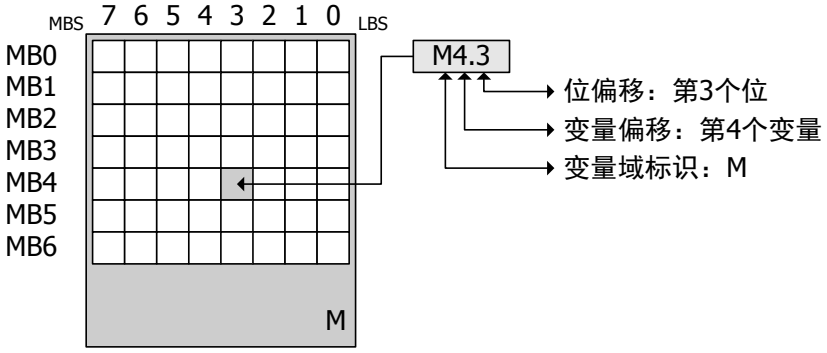
对于 I、Q 区域的位变量，PLC 系统加入了强制功能。PLC 系统为 I、Q 这两个区域都分配了强制、强制值映像。强制映像记录对应位变量是否被强制。强制值映像纪录对应位变量被强制的值。

由于 EC20/EC20 系统中通用定时器和计数器都是 16 位的，故定时器区域 T 和计数器区域 C 都只能以字的方式访问，同时变量偏移也是以字为单位。

变量使用

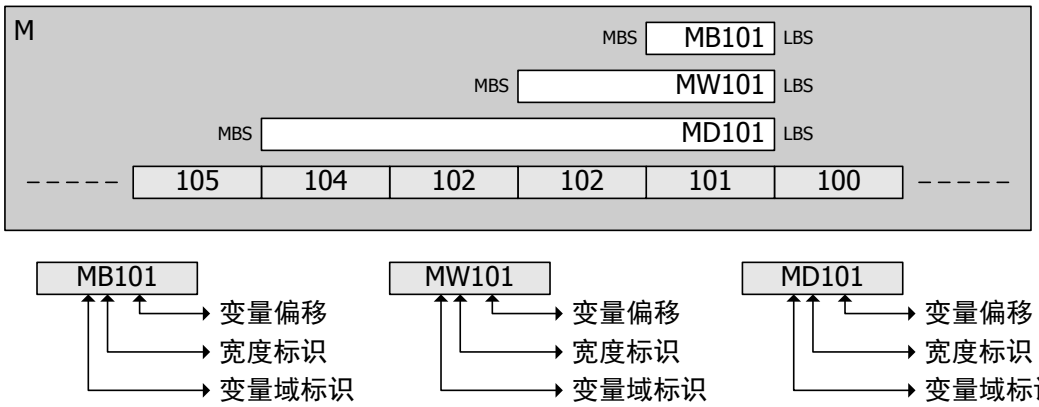
直接寻址

直接寻址直接指定变量所在的内存区域、宽度和位置。例如 MW100 表示变量在内存中的普通内存 M 区域，宽度为字（两个字节），位置是从 M 区域的第 100 个字节开始。直接寻址内存中的一个位，需要给出内存的变量域标识、变量偏移、和一个带“.”号的位偏移。例如：



直接寻址内存中的一个字节、字、双字，需要给出内存的变量域标识、宽度标识、变量偏移。在 EC20/EC30 中，宽度标识“B”表示宽度为字节（8bit），宽度标识“W”表示宽度

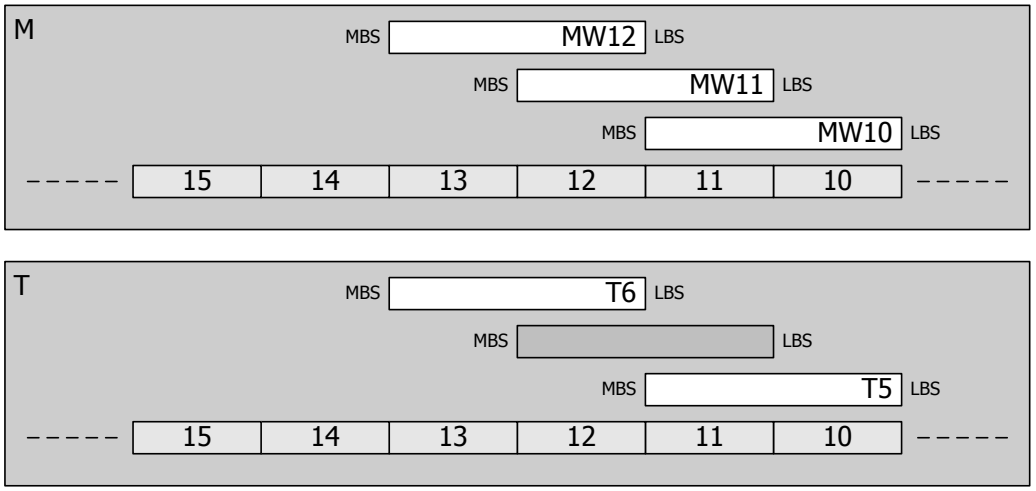
为字（16bit），宽度标识“D”表示宽度为双字（32bit）。例如：



在 EC20/EC30 中，由于定时器区域 T 和计数器区域 C 的访问只能是字，故变量域标识被省略，且变量偏移以字为单位。流程控制区域 J 的访问只能是字节，故变量域标识被省略，且变量偏移以字节为单位。高速计数器区域 HC 的访问只能是双字，故变量域标识被省略，且变量偏移以双字为单位。其余变量域的变量宽度必须通过宽度标识明确说明。

变量偏移单位

如图所示：



在变量域 M 中，由于偏移单位为字节，故 MW10 表示这个变量从 M 区域的第 10 个字节开始。又因为字节宽度是 W，故 MW10 包含 MB10 和 MB11 这两个字节。同样的，MW11 包含 MB11 和 MB12 这两个字节，MW12 包含 MB12 和 MB13 这两个字节。

在变量域 T 中，由于偏移单位为字，故 T5 表示这个变量从区域的第 5 个字（即第 10 字节和第 11 字节组成的字）。同样的，T6 表示这个变量区域的第 6 个字（即第 12 字节和第 13 字节组成的字）。由第 11 字节和 12 字节组成的字不能被寻址。

字节对齐单位

由于某些 CPU 对内存的访问有字节对齐的要求。例如载入字时需要字地址是字对齐的（地址能被 2 整除），在入双字时需要双字地址是双字对齐的（地址能被 4 整除）。对于非地址对齐的变量进行操作，需要做特殊处理。

EC20/EC30 的变量区域访问一般没有字节对齐的要求。例如 MW1 是合法的字变量，MD1、MD2、MD3 都是合法的双字变量。由于变量域 T 和变量域 C 偏移单位是字，只能进行以字为单位的访问，因此 T 和 C 上的访问永远都是字对齐的。

间接寻址

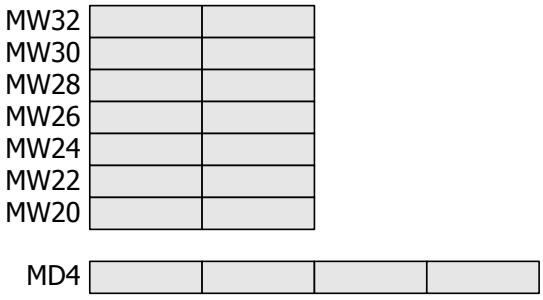
通过直接寻址我们知道，一个变量的变量名直接指定变量所在的内存区域、宽度和位置。这些信息最终会被编译成二进制的描述存储在 PLC 的程序页空间内。然而在一些场合，直接寻址是不太方便的。例如需要操作一大片数据时，指名道姓的对数据中所有的元素进行一个个重复的操作会让程序看上去很繁琐且容易出错。现在有一种方法，将某个变量（例如变量 A）的内存区域、宽度和位置信息保存到另一个变量中去（例如变量 B）。需要寻址变量 A 的时候，先寻址变量 B。找到变量 B 后读取变量 B，然后根据变量 B 中 A 的信息再去寻址变量 A。这种方法就是所谓的间接寻址。为什么不直接寻址 A 而是通过 B 去寻址 A 呢？其实好处就在这里：变量 B 是可以操作的。若把变量 B 加 1，那么变量 B 就指向了变量 A 的下一个变量。同样，若把变量 B 减 1，那么变量 B 就指向了变量 A 的上一个变量。在需要对大量连续变量进行操作时，把存放信息的变量加加减减，就能寻址所有的变量了。

在 EC20/EC30 中，变量的信息长度为双字（32bit）。若需要提取某变量的信息，必须用双字来存储。提取某变量的信息时，必须在被提取信息的变量前加上“&”号。同时被提取信息的变量必须是字节宽度。

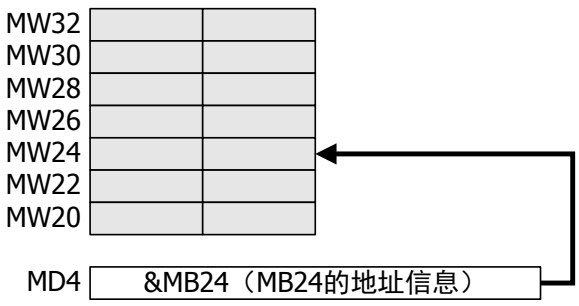
在 EC20/EC30 中，只有 M、V、SM、L 区域中的字节变量信息可以被提取。只有 M、V、L 区域中的双字变量可以用来存储变量信息。

下面是一个间接寻址的例子：

1. 未执行任何操作前：

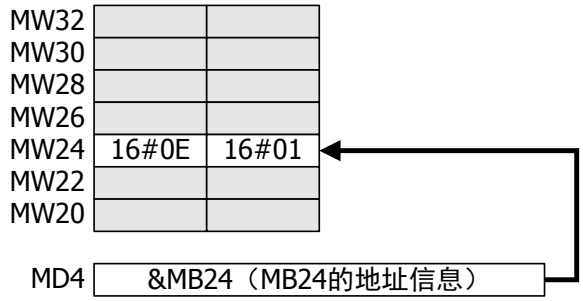


2. 执行指令：MOVD &MB24, MD4



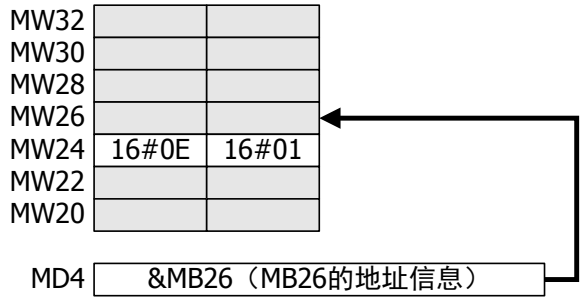
此时 MB24 的地址信息变存储在变量 MD4 中。

3. 执行指令：MOVW 16#0E01, *MD4



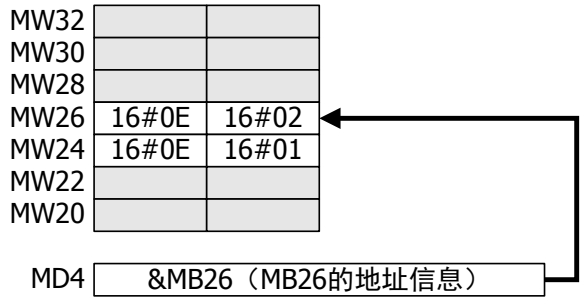
*MD4 表示间接寻址。由于 MD4 存储的是 MB24 的地址信息，又因为数据移动指令为字移动，故执行指令后 MW24 的值等于 16#0E01。

4. 执行指令：+D 2, MD4



将 MD4 保存的地址信息加 2 后，MD4 中的地址信息变为 MD26。

5. 执行指令：MOVW 16#0E02, *MD4



*MD4 表示间接寻址。由于 MD4 存储的是 MB26 的地址信息，又因为数据移动指令为字移动，故执行指令后 MW26 的值等于 16#0E02。

数据类型

基本数据类型	数据宽度（位）	说明	范围
BOOL	1	布尔	0 ~ 1
BYTE	8	字节	16#00 ~ 16#FF
WORD	16	字	16#0000 ~ 16#FFFF
DWORD	32	双字	16#00000000 ~ 16#FFFFFFFF
SINT	8	单整数	-128 ~ 127
INT	16	整数	-32768 ~ 32767
DINT	32	双整数	-2147483648 ~ 2147483647
USINT	8	无符号单整数	0 ~ 255

UINT	16	无符号整数	0 ~ 65535
UDINT	32	无符号双整数	0 ~ 4294967295