评测MAXQ指令系统和其它RISC竞争者

本文比较了MAXQ与其它竞争微控制器的指令系统，包括PIC16CXXX(中档器件)、AVR和MSP430。并以表格的方式列出了各种指令系统和架构的优势和弱点。我们将利用选定的算法和操作代码来判断不同系统的代码密度和代码性能。最后一节重点介绍各实例代码的MIPS(百万指令每秒)/mA比率。

MAXQ指令系统概览

MAXQ指令系统建立在传送-触发概念之上。指令字简单地由源和目的操作数构成。这些源和目的操作数能够表示物理寄存器，通过编码还可以间接访问数据存储器、堆栈存储器和工作累加器，和/或用于隐含地触发硬件操作。有关MAXQ的传送-触发架构，在本期前一篇文章中有进一步的介绍。特定MAXQ器件的源和目的编码在该器件相关的MAXQ用户指南中有详细说明。尽管有些源和目的编码只适用于特定器件，例如用于外围硬件功能的编码，还是有一些固定的编码为不同器件所共有，它们构成了基本的MAXQ指令系统。图1给出了MAXQ指令字和指令系统助记符。

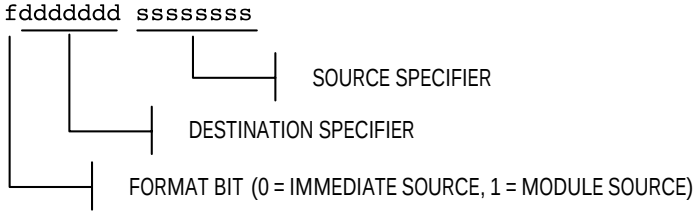


图1. MAXQ指令字所表达的源-目的传送概念产生了一个小巧且强有力的指令集。

MNEMONIC	DESCRIPTION	MNEMONIC	DESCRIPTION
BIT MANIPULATION		LOGICAL	
MOVE C, #0/#1	Clear/Set Carry	AND	Logical AND
CPL C	Complement Carry	OR	Logical OR
AND Acc.<b>	Logical AND Carry with Accumulator Bit	XOR	Logical XOR
OR Acc.<b>	Logical OR Carry with Accumulator Bit	CPL,NEG	One's, Two's Complement
XOR Acc.<b>	Logical XOR Carry with Accumulator Bit	SLA,SLA2, SLA4	Shift Left Arithmetically 1,2,4
MOVE C, Acc.<b>	Move Accumulator Bit to Carry	SRA,SRA2,SRA4	Shift Right Arithmetically 1,2,4
MOVE Acc.<b>,C	Move Carry to Accumulator Bit	SR	Logical Shift Right
MOVE C, src.<b>	Move Register Bit to Carry	RR,RRC	Rotate Right Carry (Ex/In) clusive
MOVE dst.<b>, #0/#1	Clear/Set Register Bit	RL,RLC	Rotate Left Carry (Ex/In) clusive
MATH		DATA TRANSFER	
ADD, ADDC	Add Carry (Ex/In) clusive	XCHN	Exchange Accumulator data nibbles
SUB, SUBB	Subtract Carry (Ex/In) clusive	XCH *(MAXQ20)*	Exchange Accumulator data bytes
FLOW CONTROL AND BRANCHING		MOVE dst, src	Move source to destination
JUMP {C/NC/Z/NZ/E/NE/S}	Jumps - unconditional or conditional, relative or absolute	PUSH/POP	Push/Pop stack
DJNZ LC[n], src	Decrement Counter, Jump Not Zero	POPI	Pop stack and enable interrupts (INS≤0)
CALL	Call – relative or absolute	Other	
RET {C/NC/Z/NZ/S}	Return – unconditional or conditional	NOP	No Operation
RETI {C/NC/Z/NZ/S}	Return from Interrupt – unconditional or conditional	CMP	Compare with Accumulator

...指令集和器件架构(指令周期、存储器模型、寄存器组、寻址模式等等)是密不可分的，必须统一考虑。

表1. 指令系统对比

指令系统	强项	弱项
AVR	• 32个通用工作寄存器(累加器) • 数据指针是可直接寻址的工作寄存器的一部分；易于对指针的高/低字节进行屏蔽和位操作 • 可从指针+偏移量(0至63字节偏移量)读取数据 • 堆栈仅受限于内部RAM(除了90S1200，它没有RAM，堆栈深度=3) • 单周期操作 • 相对跳转至±2k(两周期) • 所有AVR具有数据EEPROM • 显式指令设置/清除各个状态寄存器标志；大量的位操作指令 • 独立的中断矢量	• 通过流水线取指 • 超出32个寄存器时，装载(LD)/存储(ST)开销比变为一个因数：LD/ST @X, Y, Z = 两周期 • LPM=3周期 • 缩减了对于字符操作的支持/范围(没有ADDC、EOR1；仅CPI、ORI、ANDI、SUBI、SBCI、LDI工作在R16–R31上) • 没有排除进位的循环指令 • 条件跳转范围仅有+63/-64(两周期) • CALL/RET/RETI=四周期
PIC16CXXX	• 源、目的位被编入ALU操作内 • 直接数据访问(符号寻址模式)可产生高密度代码，并有利于数据的覆盖	• 四时钟内核运行速度较差 • 通过流水线取指 • 访问上区数据存储器需要翻页(RP1:0分区选择) • 间接数据访问需要INDF特殊功能寄存器 • 不能直接装载W(累加器) • 无ADDC、SUBB • 堆栈深度=8 • 无相对跳转/分支，仅有绝对(CALL、GOTO)或条件跳过(BTFSx) • RETLW读取代码存储器=浪费代码空间，且无法对代码空间实施CRC • CALL/GOTO/RET/RETFIE/RETW全部需要八个时钟周期(两个指令周期) • 单个中断向量

MAXQ对比其它指令系统

有人可能会试图拿MAXQ的指令助记符来和其他系统作对比，但应该强调，这种分析会十分困难且不公平，因为每种指令系统都是基于特定的器件资源和寻址模式而建立的。基于这个原因，指令系统和器件架构(指令周期、存储器模型、寄存器组、寻址模式等等)是密不可分的，必须统一考虑。**表1**归纳了几种拿来做比较的指令系统各自的强项和弱项。

代码实例

比较不同指令系统的最佳方法是定义一系列任务，并在不同体系下编写代码来执行这些任务。随后诸节介绍这些用于执行的任务，并归纳了各指令系统的代码密度和性能表现。第一个例程的实例代码包含在本文中，但后面的例程仅在图表和正文中对其性能加以归纳。对应于各统计结果的代码例程如果需要可向Dallas Semiconductor索取。

表1. 指令系统对比(续)

指令系统	强项	弱项
MSP430	• 操作码内含有多种源、目的寻址模式编码——可产生高密度代码 • 16 位内部数据通道 • 内部存储器可以字或字节方式寻址 • 常数发生器 (CG) 产生 -1、0、1、2、4、8 • 单周期操作 • 堆栈仅受限于内部 RAM • 条件/相对跳转目的地范围 = ±512 (两周期) • 独立的中断向量，自动清除单源标志	• Von Neumann 存储器映射 + 精细的寻址模式 = 多个执行周期。只有那些独占式处理 Rn 的指令为单周期。外设寄存器访问 = 3 至 6 个周期 • CG 不支持的文字需要额外的字 • 目的操作数不能由寄存器间址或寄存器间址自动递增 • 寄存器间址不支持自动递减 • 符号寻址限制了例程代码的再利用
MAXQ	• 系统和外设寄存器可作为同一逻辑存储空间内的源和目的进行访问，加速了数据的传送 • 单周期操作，且无流水线 • 单周期条件跳转 (+127/-128) 或两周期绝对跳转 (0−65,535) • 单周期 CALL/RET/RETI • 自动递减的循环计数器省去了通常用于处理计数的开销 • 支持自动递增/递减功能的三个数据指针。一个数据指针 (FP) 支持基指针+偏移量寻址方式 (即 BP[Offs]) • 对于累加器 (工作寄存器) 文件进行自动递增/递减/取模控制 • 每个数据指针具有可选的字或字节访问模式 • 可加前缀的操作码便于简单实现指令集的扩充或增强	• 活动累加器始终是 ALU 操作的隐含目的操作数 • 单端口，同步，SRAM 数据存储器要求数据指针在使用前首先要激活 (选定) • 默认堆栈深度 = 16，不过，利用数据指针硬件资源非常容易在数据存储器中实现软堆栈

存储器拷贝 (*MemCpy64*)

系统和外设寄存器可作为同一逻辑存储空间内的源和目的进行访问，加速了数据的传送。

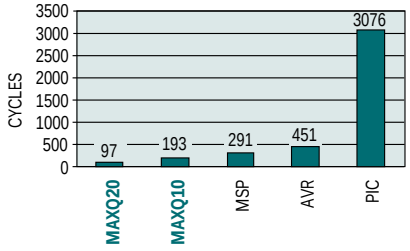
存储器拷贝例程表现了微控制器间接操作数据存储器块的能力。任务是从一个数据存储器源地址拷贝 64 字节数据到一个与源不相重叠的数据存储器目的地。随后几页给出了用于各种微控制器的代码，并以图表方式总结了各种代码执行拷贝操作所用的周期数和字节数。这些例程中均假定在拷贝操作之前已设定了指针和字节数，并且将要拷贝的字节在存储器中是字对齐的，以便使用 MSP430 和 MAXQ20 的字访问模式。

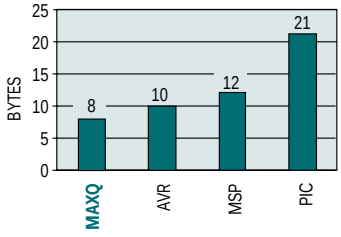
```
;====================================AVR=====================================
; ramsize=r16         ;size of block to be copied
; Z-pointer=r30:r31   ;src pointer
; Y-pointer=r28:r29   ;dst pointer
; USES:
; ramtemp=r1          ;temporary storage register
loop:                           ; cycles
       ld     ramtemp,Z+        ; 2 @src => temp
       st     Y+,ramtemp        ; 2 temp => @dst
       dec    ramsize           ; 1
       brne   loop              ; 2/1
       ret                      ;  4/5
                                ;---------
                                ;(7*bytecount) + return –1(last brne isn't taken).
; WORD COUNT = 5 ; CYCLE COUNT = 451


;====================================MAXQ10====================================
; DP[0] ; src pointer (default WBS0=0)
; DP[1] ; (dst-1) pointer (default WBS1=0)
; LC[0] ; byte count (Loop Counter)
loop:                                   ;words & cycles
       move   DP[0], DP[0]              ; 1  implicit DP[0] pointer selection
       move   @++DP[1],@DP[0]++         ; 1
       djnz   LC[0], loop               ; 1
       ret                              ; 1
                                        ;----------
                                        ; 4 / (3*bytecount) +1
; WORD COUNT = 4 ; CYCLE COUNT = 193


;====================================MAXQ20====================================
; Assuming bytes are word aligned (like MSP430 code) for comparison
; DP[0] ; src pointer (default WBS0=1)
; DP[1] ; (dst-1) pointer (default WBS1=1)
; LC[0] ; byte count  (Loop Counter)
loop:                                   ;words/cycles
       move   DP[0], DP[0]              ; 1  implicit DP[0] pointer selection
       move   @++DP[1],@DP[0]++         ; 1
       djnz   LC[0], loop               ; 1
       ret                              ; 1
                                        ;----------
                                        ; 4 / (3*bytecount/2) +1
; WORD COUNT = 4 ; CYCLE COUNT = 97


;====================================MSP430====================================
; MSP430 has a 16-bit data bus
; assuming bytes are word aligned, only requires (blocksize/2 transfers).
; R4   ;src pointer
; R5   ;dst pointer
; R6   ;size of block to copy
loop:                           ;words/cycles
       mov    @R4+, 0(R5)       ;2 / 5  @src++ => dst
       add    #2, R5            ;1 / 1  const generator makes this 1/1
       decd.b R6                ;1 / 1  really sub #2, R6
       jz     loop              ;1 / 2
       ret                      ;1 / 3
                                ;----------
                                ;6 / (9*(bytecount/2)) + return
; WORD COUNT = 6 ; CYCLE COUNT = 291
;==================================PIC16CXXX===================================
; a     ; src pointer base
; b    ; dst pointer base
; i    ; byte count held in reg file
; USES:
; temp  ; temp data storage
loop:                                   ; cycles
        decf    i, W                    ; 1 i-- => W
        addlw   a                       ; 1 (a+i--) => W starting at end
        movwf   FSR                     ; 1 W => FSR
        movfw   INDF                    ; 1 W <= @FSR get data
        movwf   temp                    ; 1 W => temp
```

```
        movlw   (b-a)               ; 1 diff in dest-src
        addwf   FSR, F              ; 1 (b+i--) => W
        movfw   temp                ; 1 temp => W
        movwf   INDF                ; 1 W => @FSR store data
        decfsz  i, F                ; 2/1 i--
        goto    loop                ; 2
        return                      ;   2
                                    ;----------
                                    ;11 / (12*bytecount) +1 (ret instead of
goto, +1 on decfsz)
; WORD COUNT = 12 ; CYCLE COUNT = 769 (*4clks/inst cycle = 3076)
```

MAXQ 在存储器拷贝实例中所表现出的优势，对于需要频繁地在数据存储器中进行输入/输出缓冲的应用来讲，可转化为类似的好处。

MemCpy64 周期/字节数比较





MAXQ 器件具有最佳的代码密度，并且在运行速度方面具有显著的优势。MAXQ10 执行拷贝操作的速度慢于 MAXQ20，是因为它的数据指针采用的是默认的字节访问模式。对于MAXQ10 应用，如果运行速度比代码密度更为重要，并且将要拷贝的存储器是字对齐的(正如我们在 MSP430 和 MAXQ20 实例中所假定的)，它也可以采用字访问模式的源和目的数据指针。启用字模式可使 MAXQ10 的拷贝循环缩短一半，但需要额外的指令来使能/禁止字访问模式。MAXQ 器件较之竞争者所表现出的压倒性优势归功于下列结构上的优势:

1) 无流水线——程序分支不会导致其它器件所固有的指令预取开销。

2) 自动递减的循环计数器——节省了软件开销。

3) Harvard 存储器映射——程序和数据不必共享同一物理空间，允许同时取指和访问数据。

4) 后递增/递减的间接目的指针——简化并加速了目的指针的步进。这是 MSP430 的一个弱项，它采用 0(R5) 表示@R5，然后，必须在后续指令中步进该目的指针。

MAXQ 在存储器拷贝实例中所表现出的优势，对于需要频繁地在数据存储器中进行输入/输出缓冲的应用来讲，可转化为类似的好处。就性能而言，最接近的竞争者是 MSP430。作为一个用数据存储器进行缓冲的实例，假定我们的 MSP430 配备了一个具有 16 位输出寄存器的 ADC 外设。要完成由外设输出寄存器到数据存储器的数据转移，并递增指针以便为下一个 ADC 输出采样做好准备，这项任务可由以下代码完成:

```
                                ; words/cycles
mov.w   &ADAT,0(R14)            ; 3 / 6       Store output word
        incd.w  R14             ; 1 / 1       Increment pointer
                                ; 4 / 7
```

同样的转移操作，对于 MAXQ20 可完成如下:

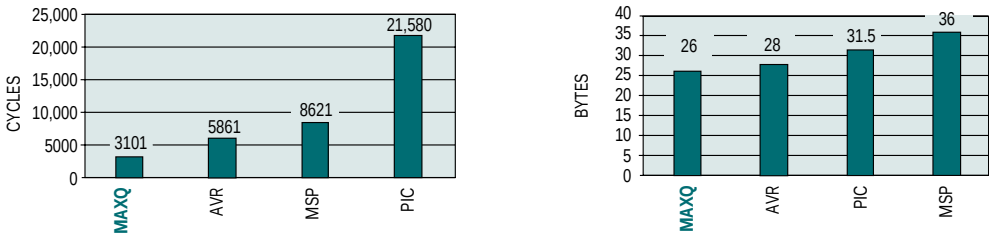
```
move    @DP[0]++, ADCOUT        ; 1 / 1
```

气泡法排序程序不仅可以表现高效访问数据存储器的能力，同时也对数据字节执行算术和/或比较操作，并有条件地对数据字节重新排序。

气泡法排序 (*BubbleSort*)

气泡法排序程序不仅可以表现高效访问数据存储器的能力，同时也对数据字节执行算术和/或比较操作，并有条件地对数据字节重新排序。本程序代码对32个数据存储器字节进行排序，使它们按升序或降序排列。本例中的周期数，是在假定相邻字节比较中，有近一半的机会发生重新排序的情况下得到的。下图归纳了各种微控制器执行排序操作所需的字节数和周期数。

BubbleSort 周期/字节数比较

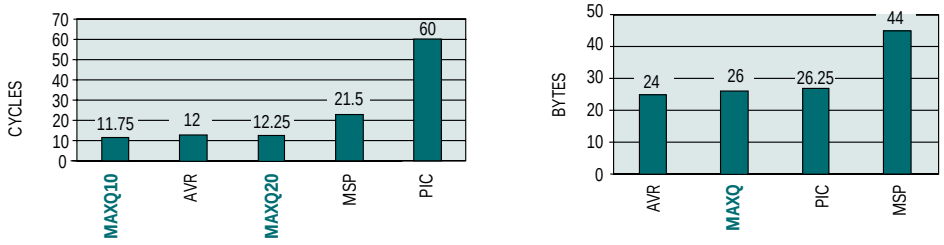


再一次，MAXQ器件提供了最佳的代码密度，并且在运行速度方面远远胜出。MAXQ的优势来源于其架构上的优越性，正如我们在存储器拷贝实例中所讨论的。

十六进制到 ASCII 转换 (*Hex2Asc*)

该转换例程考察微控制器的算术和逻辑运算能力。同时它还可以考察微控制器对于字符数据的支持，测试它们从单个文本字节中转换并解析数据的能力。测得的周期数为平均值，并假定每个半字节均为16个十六进制值之一(0至9，A至F)。下图归纳了各种微控制器执行转换操作所需的字节数和周期数。

Hex2Asc 周期/字节数比较



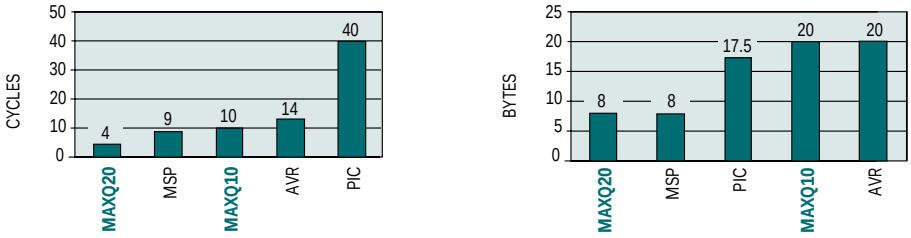
在本测试中，AVR的代码量少了一个字，原因是它的工作寄存器可直接寻址，而MAXQ的最高效算法需要更新累加器指针。MSP因缺乏半字节操作指令，而不被常数发生器支持的字符数据(#nnnnh)又必须编码为单独的字，因而代码密度受到了影响。在性能方面，MAXQ和Atmel AVR取得了相近的成绩，而其它器件落在了后面。MSP430的性能受到了其过多操作代码的拖累。

算术右移 2 位 (*ShRight*)

该例程演示了微控制器支持16位字存储器访问和ALU操作的能力。操作要求是对位于数据存储器内的一个16位字进行算术移位(即保留高位)。假定数据字位于数据存储器的第一个256字节块内，并且是字对齐的，可以由那些具备字访问能力的控制器按字寻址。下图归纳了各种微控制器执行移位操作所需的字节数和周期数。

两种支持16位ALU操作的微控制器，MAXQ20和MSP430，在代码密度方面具有显著优势。所有的8位机都需要两倍以上数量的码字来执行相同的算术移位操作MAXQ20具有最佳性能，而仅支持8位ALU操作的MAXQ10性能接近16位的MSP430。

ShRight 周期/字节数比较

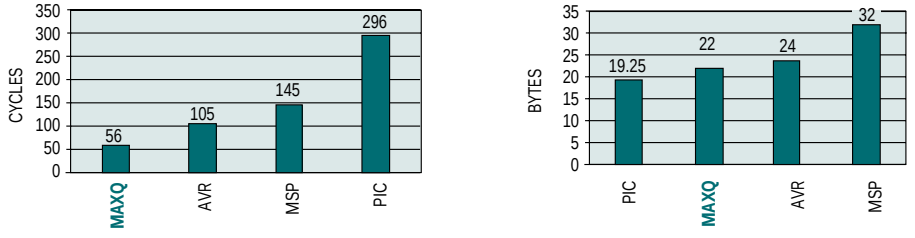


MAXQ20和MSP430具有较高的代码密度，因为它们能够比8位机更高效地操纵16位数据存储器。不过，它们的操作方式略有不同。MAXQ20将准备移位的16位字传送给工作寄存器(累加器)，在那里，它可执行多位算术移位操作。MSP430则是利用寄存器间址模式(RRA@R5)执行单位算术移位操作，并不显式地在存储器内传送数据字。除了具备更高性能，MAXQ20还可提供和MSP430同等或更佳的代码密度，特别是当16位字的算术移位可以利用一种多位算术移位操作码完成时(SRA2，SRA4，SLA2，SLA4)。

位控端口引脚(*BitBang*)

本例用于测试指令系统通过直接位操作或移位/循环方式分解数据字节，并将分解出的独立位元发送到某个端口引脚(“位控”)的能力。端口引脚的输出可分别代表时钟和数据，并要求数据必须在时钟的上升沿有效。由于代码中要直接操纵端口引脚，该项测试也展示了访问I/O端口寄存器的能力。下图归纳了各种微控制器执行端口位控操作所需的字节数和周期数。

BitBang 周期/字节数比较



MAXQ又一次展示出明显的性能优势。PIC的性能由于4周期内核的架构而受到限制(亦如其它例程)。MSP430性能较差，主要归因于其Von Neumann存储器架构和必须使用绝对寻址访问端口输出寄存器。

BitBang例程测试指令体系通过直接位操作或移位/循环方式分解数据字节，并将分解出的独立位元发送到某个端口引脚(“位控”)的能力。

至于代码密度，MAXQ和PIC具有相同字数。不过在RISC机中PIC还是胜出MAXQ一筹，因为它是14位程序字而MAXQ是16位程序字。MSP430的代码密度较低，主要是因为当它以绝对寻址模式(即&register)访问外设寄存器或处理无法用常数发生器减省的字符数据时(例如#3h)时，必须至少用去两个字。

MSP430访问外设寄存器的方式有必要作进一步的说明。微控制器的主要职责之一是以多种方式与外部世界接口。因此它必须控制、监视和处理发生在I/O引脚上的事件。如果微控制器只包含了极少的外围硬件模块，那么这项负担就留给了软件。软件如果要做一些有实际意义

在考虑工作电流消耗时，MIPS/mA 性能提供了一个用以评估微控制器代码效率的简单手段。

的事情，它就必须对端口引脚进行读和写。对于 MSP430，这些端口寄存器位于外设寄存器空间，必须采用绝对访问模式。现在，考虑一个具有丰富的“智能”外设的微控制器。那么，在使用这些片上的专用硬件执行必要功能的过程中，无疑将会有更多的外设寄存器需要配置、控制和访问。在 MSP430 中，这些寄存器位于外设寄存器空间，它们要求采用绝对寻址模式访问。最终结果是，MSP430 的绝对寻址模式对于代码密度和性能的妨害就无可避免。

“MIPS/mA”性能

功率消耗常常是选择处理器或内核架构的重要因素。一个系统的总体功耗与许多因素有关，例如电源电压和工作频率，以及它在任何可能的情况下使用低功耗模式的能力。降低电源电压和/或工作频率，同时频繁使用低功耗模式，可以大幅度降低总体系统的功耗。然而，对于一个给定的微控制器，最低电源电压很大程度上依赖于器件的工艺技术，工作频率的降低和低功耗模式的使用则主要取决于具体应用的要求，可由系统设计者来决定。在考虑工作电流消耗时，MIPS/mA 性能提供了一个用以评估微控制器代码效率的简单手段。在比较不同器件的 MIPS/mA 性能时，要获得有意义的结果就应该选择共同的电源电压。在后面的比较中，假定电源电压为 3V。要比较各指令系统之间 (即 AVR、MSP430、PIC16、MAXQ)的差异和效率，还有必要为每一种代码例程提供单独的 MIPS/mA 比例。

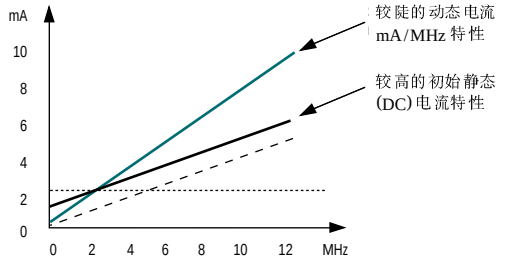


图2. 该工作电流-MHz 图例说明了静态电流和动态电流的增加效应。

为确定 MIPS/mA 比例中的“mA”部分，我们来检查一下器件的数据手册。大多数微控制器提供商规定了器件在最大工作频率下的典型和最大工作电流。假定静态 (DC) 电流非常小，利用这些数据，我们可以估算出一个近似的 mA/MHz 值，并由它导出任意时钟频率下的工作电流。如有供应商提供的工作电流随温度/频率变化特性数据的话，我们可以很好地量化特定环境条件下的 mA/MHz 比。否则，只能依赖于几个分离的数据点和静态电流很小的假设。静态 (DC) 电流的增加改变了 mA-MHz 特性曲线的起点，因而限制了系统设计者通过降低时钟频率所能得到的好处 (降低动态电流)。图2 给出了一个工作电流-MHz 曲线实例。表2 比较了不同内核的 mA/MHz 数据，并列出了这些信息的来源。突出显示的各种控制器 mA/MHz 数据将被用于后面的计算。

MIPS/mA 中的“MIPS”部分用来衡量性能方面的差异。作为出发点，我们将从一个简单的 MIPS 方程开始，如图3 所示。

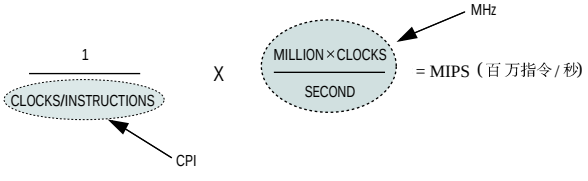


图3. MAXQ 架构几乎所有的指令都在一个时钟内执行，因而赢得了很高的 MIPS 性能比例。

在评估一个给定架构的 MIPS 时，每条指令的时钟数 (CPI)需要高度重视。有些指令系统，例如 Microchip PIC，每个指令周期要求多个时钟。此外，有些架构常常需要多个指令周期来完成某些指令，或需要一些周期来重装指令流水线 (跳转/分支指令)。在比较不同架构时，平均 MIPS 常常远低于峰值性能 (MIPS)，其差距取决于指令的组合。

表2. 不同内核 mA/MHz 数比较

器件	典型 mA/MHz	最大 mA/MHz	来源
PIC16C55X	0.7	1.25	PIC16C55X data sheet: DC Table 10.1, D010 (V_{CC} = 3V, 2MHz); XT or RC
PIC16C62X	0.7	1.25	PIC16C62X data sheet: DC Table 12.1, D010 (V_{CC} = 3V, 2MHz); XT or RC
PIC16LC71	0.35	0.625	PIC16C71X data sheet: DC Table 15.2, D010 (V_{CC} = 3V, 4MHz); XT or RC
PIC16F62X	0.15	0.175	PIC16F62X data sheet: DC Table 17.1, D010 (V_{CC} = 3V, 4MHz)
PIC16LF870/1	0.15	0.5	PIC16F870/1 data sheet: DC Table 14.1, D010 (V_{CC} = 3V, 4MHz); XT or RC
AT90S1200	0.33	0.75	AT90S1200 data sheet: EC Table (3V, 4MHz), Figure 38, 4mA/12MHz (typ)
AT90S2313	0.50	0.75	AT90S2313 data sheet: EC Table (3V, 4MHz), Figure 57, 7.5mA/15MHz (typ)
MSP430F1101	0.30	0.35	MSP430x11x1 data sheet: DC specs IccActive (V_{CC} = 3V, FMCLK = 1MHz)
MSP430C11X1	0.24	0.30	MSP430x11x1 data sheet: DC specs IccActive (V_{CC} = 3V, FMCLK = 1MHz)
MSP430Fx12x	0.30	0.35	MSP430x12x data sheet: DC specs (V_{CC} = 3V, FMCLK = 1MHz, FACLK = 32kHz)
MAXQ10	0.30		**Simulations**
MAXQ20	0.30		**Simulations**

为获得一个更为有用的指标，并产生一个帮助我们到达 MIPS/mA 最终目标的数值，我们用 MIPS 除以 MHz。MIPS/MHz 比例可解释为平均每个时钟周期可执行的指令数(对于指定的代码实例)。利用 MIPS/MHz 数和前面计算出的 mA/MHz 数，便可得到 MIPS/mA 数。下面的表格分别显示了前面比较所用的各种代码例程的 MIPS/MHz 和 MIPS/mA 数。

表3. 选定代码算法的 MIPS/MHz 和 MIPS/mA 比较

CORE	MIPS/MHz					
	MemCpy64	**BubbleSort**	**Hex2Asc**	**ShRight**	**BitBang**	***Peak***
MAXQ10	**1.00**	**0.99**	**1.00**	**1.00**	**1.00**	*1*
MAXQ20	**1.00**	**0.99**	**1.00**	**1.00**	**1.00**	*1*
PIC	0.23	0.20	0.23	0.23	0.21	*0.25*
MSP	0.44	0.39	0.64	0.33	0.38	*1*
AVR	0.57	0.62	0.90	0.71	0.61	*1*

CORE	MIPS/mA				
	MemCpy64	**BubbleSort**	**Hex2Asc**	**ShRight**	**BitBang**
MAXQ10	**3.33**	**3.30**	**3.33**	**3.33**	**3.33**
MAXQ20	**3.33**	**3.30**	**3.33**	**3.33**	**3.33**
PIC	1.53	1.35	1.53	1.50	1.40
MSP	1.85	1.62	2.66	1.39	1.55
AVR	1.71	1.86	2.69	2.14	1.83

为了更进一步的分析，我们还应该用 MIPS/mA 比除以给定代码所需要执行的实际指令数，以便衡量不同内核架构和指令系统效率之间的差距。作此额外计算的理由是，运行三条单周期指令(具有最高的 MIPS/MHz 比，等于 1)其实不如一条 3 周期指令(MIPS/MHz 比 = 0.33)。但是，MIPS/mA 比却形成了强烈反差。事实上，如果执行任务相同的话，大多数人更喜欢

MIPS/MHz 比例可解释为平均每个时钟周期可执行的指令数(对于指定的代码实例)。

单条指令，而非三条指令。将 MIPS/mA 比除以运行指令数，我们就得到了给定微控制器采用一定的指令组合执行特定任务时的 MIPS/mA 比。最终结果被归一化至表现最优者，并在下表中给出。

表 4. 归一化 MIPS/mA 数值比较

CORE	NORMALIZED (MIPS/mA)				
	MemCpy64	BubbleSort	Hex2Asc	ShRight	BitBang
MAXQ10	0.50	**1.00**	**1.00**	0.40	**1.00**
MAXQ20	**1.00**	**1.00**	0.96	**1.00**	**1.00**
PIC	0.06	0.29	0.39	0.20	0.38
MSP	0.42	0.45	0.68	0.56	0.48
AVR	0.19	0.48	0.88	0.26	0.48

结论

归一化的“MIPS/mA”性能为我们提供了一个相对的性能/电流比率，可用来比较具有不同架构、指令系统和功耗性能的微控制器。较高的归一化“MIPS/mA”比通常可提供下面两方面好处中的一种或全部：(1)有可能降低系统时钟频率，以及(2)处于低功耗或休眠模式的时间有可能延长。这两种可能性都可用于降低系统的总体功耗。另一方面来讲，在给定的电流/功耗预算内，有可能实现更高的系统总体性能。无论何种好处，MAXQ 架构所具有的高 MIPS/mA 比都是一个值得信赖的高效能典范。