Assembly Language, Floating Point, PWM

The ARM Cortex M0 processor supports only the thumb2 assembly language instruction set. This instruction set consists of fifty 16-bit instructions and six 32-bit instructions.

The general syntax for the assembly language in Keil is:

```
label mnemonic operand1, operand2, ...; comments
```

Note that there is no colon separating the label from the mnemonic and you can put the label on a separate line by itself. The label is optional. The first operand is the destination operand if there are multiple operands. Everything after a semicolon is a comment until the next line.

**In Line and Embedded Assembly**

To use inline assembly code you must use the `__asm` directive. This allows you to place assembly code inside your c-code as in:

```
__asm {
    movs r1, r2;
    movs r3, r2;
}
```

Inline assembler is not available for the ARM Cortex M0 processor but is available on processors that support the full 32-bit instruction set.

Embedded assembly is more useful. Embedded assembly places an assembly function inside a C program and can be used with the ARM Cortex M0 processor.

Example 1 shows how to use an assembly function along with C code to toggle bit P0.7 as fast as possible. The assembly function creates an empty loop as a software delay. Example 2 uses an assembly function to set or clear a bit on port 0.

Embedded assembly is occasionally useful for sections of code that require lots of bit manipulation or that may require the fastest possible speed. For example the ARM processor has a barrel shifter that is available in hardware which is not used by the C compiler. It could be accessed by way of embedded assembly code.

---

**Example 1**

Toggles PIO0_7 using an assembly function

```
// Outputs pulse at about 1.3 MHz without the for loop delays
#define IOCON_PIO0_7 (*((volatile unsigned long *) 0x40044050))   // Pin control register
#define GPIO0DATA (*((volatile unsigned long *) 0x50003FFC))   // Port 0 data register
#define GPIO0DIR (*((volatile unsigned long *) 0x50008000))   // Port 0 data direction register
#define SYSAHBCLKCTRL (*((volatile unsigned long *) 0x40048080))   // System AHB clock control

__asm void Delay(int x) {
    movs r1, #0;       // R1 = 0
    loop
        addrs r1, r1, #1;  // incr R1
        cmp r0, r1;        // Compare R1 to x in R0
        bgt loop;          // Branch greater than to loop
    bx lr;             // Return
}
```
int main()
{
    SYSAHBCLKCTRL |= (1 << 6); //Enable clock for GPIO
    GPIO0DIR |= (1 << 7);      //Pin direction to output
    while(1)
    {
        GPIO0DATA |= (1 << 7);
        Delay(5000);
        GPIO0DATA &= ~(1 << 7);
        Delay(5000);
    }
}

Note that an s has been added to many assembler mnemonics as in
adds r1, r2, #1. Most ARM instructions are available with or without the s
where the instruction without the s does not alter the flags while the instruction with
the s does. For our particular compiler/assembler, this is not an option and the added
s is necessary.

Example 2
This example uses an assembler function to write to a specific bit position on port 0.

/*ASMWriteBit.c
Accepts a bit value and a bit position and writes the value
to the bit position on P0. Program sequentially changes all
of the bits on P0 to 1 followed by all to 0. Note that PIO0_4
PIO0_5 are used for I2C functions are are open drain only.
To see these go from 0 to 1 you need a pull up resistor
on each pin.
The first four parameters are passed in R0 to R3. After that
parameters are passed on the stack. Any return value is
passed back in R0.
*/
#define GPIO0DATA (*((volatile unsigned long *) 0x50003FFC))  //Port 0 data register
#define GPIO0DIR (*((volatile unsigned long *) 0x50008000))   //Port 0 data direction register
__asm WriteBit(int bitPos, int value);
int main()
{
    int i, j;
    GPIO0DIR |= 0xFFF;
    while(1)
    {
        for(i=0;i<12;i++)
        {
            j = i;
            WriteBit(j, 1);
        }
        for(i=0;i<12;i++)
        {
            WriteBit(i, 0);
        }
    }
}

__asm WriteBit(int bitPos, int value)
{
    push {r0-R5};        save the registers
    movs r2, #1;         shift a 1 to the left
    lsls r2, r0;         bitPos times
    ldr r4, =0x50003FFC;  address of Port 0

4-2
Floating Point (Math) Library
To do programs in floating point arithmetic you need only declare variables of type float or
double in your C code. If you declare a float variable it will be 32-bits long and have a range of
about $0.34 \times 10^{38}$ to $+3.4 \times 10^{38}$. The float and int variable types both are 32-bits long on the
Cortex M0 processor but floating point numbers require more processing code to manage the
exponent. Floating point arithmetic will noticeably slow down processing and it will require
more space for the libraries. The ARM Cortex M0 processor also supports the double (64-bit)
data type which has a range of about $-1.7 \times 10^{308}$ to $+1.7 \times 10^{308}$. All of the standard math
functions for floating point arithmetic are available in the math.h library.

Pulse Width Modulation (PWM)
Many of the ARM Cortex processors support an onboard D/A converter. The ARM Cortex M0
does not. (There is an A/D but no D/A). Digital to analog data conversion can be done by
filtering a signal which is pulse width modulated so that all frequencies at or above the base
frequency are cutoff.

In PWM we start with a base frequency of say 1,000Hz and each cycle we output a single pulse
whose width varies from 0% of the period to 100% of the period.

![Figure 1](image)

Pulse width modulation. In this case the signal is 50% of maximum.

The base frequency of the PWM is $f = 1/T$ so that the frequency content of a PWM signal will
have frequencies at $f$ and higher due to the base frequency. The signal information is transmitted
by the duty cycle and will be, generally, a much lower frequency signal. Thus a relatively simple
low pass filter can eliminate the frequencies at the base frequency and above and recover an
analog signal from the PWM.
A timer diagram which support PWM is shown in Figure 2. The ARM Cortex M0 has two 16-bit counters and two 32-bit counters. Any one of these counters can be used to generate the base frequency.
18.7.13 Rules for single edge controlled PWM outputs
1. All single edge controlled PWM outputs go LOW at the beginning of each PWM cycle (timer is set to zero) unless their match value is equal to zero.
2. Each PWM output will go HIGH when its match value is reached. If no match occurs (i.e. the match value is greater than the PWM cycle length), the PWM output remains continuously LOW.
3. If a match value larger than the PWM cycle length is written to the match register, and the PWM signal is HIGH already, then the PWM signal will be cleared on the next start of the next PWM cycle.
4. If a match register contains the same value as the timer reset value (the PWM cycle length), then the PWM output will be reset to LOW on the next clock tick. Therefore, the PWM output will always consist of a one clock tick wide positive pulse with a period determined by the PWM cycle length (i.e. the timer reload value).
5. If a match register is set to zero, then the PWM output will go to HIGH the first time the timer goes back to zero and will stay HIGH continuously.

Note: When the match outputs are selected to serve as PWM outputs, the timer reset (MRnR) and timer stop (MRnS) bits in the Match Control Register MCR must be set to 0 except for the match register setting the PWM cycle length. For this register, set the MRnR bit to 1 to enable the timer reset when the timer value matches the value of the corresponding match register.
#define IOCON_PIO1_9 (*((volatile unsigned long *) 0x40044038))   //Pin control register
#define SYSAHBCLKCTRL (*((volatile unsigned long *) 0x40048080))   //System AHB clock control
#define TMR16B1IR (*((volatile unsigned long *) 0x40010000))   //Timer 1 Interrupt register
#define TMR16B1TCR (*((volatile unsigned long *) 0x40010004))   //Timer 1 Timer control register
#define TMR16B1PR (*((volatile unsigned long *) 0x40010008))   //Timer 1 Prescale register
#define TMR16B1TC (*((volatile unsigned long *) 0x4001000C))   //Timer 1 Timer counter
#define TMR16B1PC (*((volatile unsigned long *) 0x40010010))   //Timer 1 Prescale counter
#define TMR16B1MCR (*((volatile unsigned long *) 0x40010014))   //Timer 1 Match Control register
#define TMR16B1MR0 (*((volatile unsigned long *) 0x40010018))   //Timer 1 Match register 0
#define TMR16B1MR1 (*((volatile unsigned long *) 0x4001001C))   //Timer 1 Match register 1
#define TMR16B1MR2 (*((volatile unsigned long *) 0x40010020))   //Timer 1 Match register 2
#define TMR16B1MR3 (*((volatile unsigned long *) 0x40010024))   //Timer 1 Match register 3
#define TMR16B1CCR (*((volatile unsigned long *) 0x40010028))   //Timer 1 Capture Control register
#define TMR16B1CR0 (*((volatile unsigned long *) 0x4001002C))   //Timer 1 Capture register 0
#define TMR16B1EMR (*((volatile unsigned long *) 0x4001003C))   //Timer 1 External match register
#define TMR16B1CTCR (*((volatile unsigned long *) 0x40010070))   //Timer 1 Count Control register
#define TMR16B1PWMC (*((volatile unsigned long *) 0x40010074))   //Timer 1 PWM Control register
#include <math.h>  //PWMSine.c
const double PI = 3.141592654;

int main()
{
    int i, j, y;
    double x;
    double incr;
    incr = 2*PI/20;

    SYSAHBCLKCTRL |= (1 << 8);  //Enable clock for 16-bit timer 1
    IOCON_PIO1_9 |= 1;       //Selects match function so Pin1_9 is PWM
    TMR16B1MR0 = 256;          //Match register value.  Pin1_9 set to 1
    TMR16B1PR = 1;           //Prescale register.  Divide P Clock by 2 = 24 MHz
    TMR16B1MCR |= (1 << 10); //Causes TimerCounter to be reset if match on MR3
    TMR16B1PWMC |= 1;        //Enable PWM on Channel 1
    TMR16B1CTCR |= 1;         //Enable TimerCounter to run
    while (1)
    {
        x = 0;
        for(i=0;i<20;i++)
        {
            x = x + incr;
            //Number loaded into MR0 can be 0 to 1023
            y = (int)(480*((sin(x) + 1))+0.5);
            TMR16B1MR0 = y;
            //for(j=0;j<100;j++);
        }
    }
}
7.4.13 IOCONPIO1_9

Remark: See Section 7.1 for part specific details.

Table 70. IOCONPIO1_9 register (IOCONPIO1_9, address 0x4004 4038) bit description

<table>
<thead>
<tr>
<th>Bit</th>
<th>Symbol</th>
<th>Value</th>
<th>Description</th>
<th>Reset value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:0</td>
<td>FUNC</td>
<td></td>
<td>Selects pin function. All other values are reserved.</td>
<td>000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x0</td>
<td>Selects function PIO1_9.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x1</td>
<td>Selects function CT16B1_MAT0.</td>
<td></td>
</tr>
<tr>
<td>4:3</td>
<td>MODE</td>
<td></td>
<td>Selects function mode (on-chip pull-up/pull-down resistor control).</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x0</td>
<td>Inactive (no pull-down/pull-up resistor enabled).</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x1</td>
<td>Pull-down resistor enabled.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x2</td>
<td>Pull-up resistor enabled.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x3</td>
<td>Repeater mode.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>HYS</td>
<td></td>
<td>Hysteresis.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Disable.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Enable.</td>
<td></td>
</tr>
<tr>
<td>9:6</td>
<td></td>
<td></td>
<td>Reserved</td>
<td>0011</td>
</tr>
<tr>
<td>10</td>
<td>OD</td>
<td></td>
<td>Selects pseudo open-drain mode. See Section 7.1 for part specific details.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Standard GPIO output</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Open-drain output</td>
<td></td>
</tr>
<tr>
<td>31:11</td>
<td></td>
<td></td>
<td>Reserved</td>
<td>-</td>
</tr>
<tr>
<td>Name</td>
<td>Access</td>
<td>Address Offset</td>
<td>Description</td>
<td>Reset Value</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>TMR16B1IR</td>
<td>R/W</td>
<td>0x000</td>
<td>Interrupt Register (IR). The IR can be written to clear interrupts. The IR can be read to identify which of five possible interrupt sources are pending.</td>
<td>0</td>
</tr>
<tr>
<td>TMR16B1TCR</td>
<td>R/W</td>
<td>0x004</td>
<td>Timer Control Register (TCR). The TCR is used to control the Timer Counter functions. The Timer Counter can be disabled or reset through the TCR.</td>
<td>0</td>
</tr>
<tr>
<td>TMR16B1TC</td>
<td>R/W</td>
<td>0x008</td>
<td>Timer Counter (TC). The 16-bit TC is incremented every PR+1 cycles of PCLK. The TC is controlled through the TCR.</td>
<td>0</td>
</tr>
<tr>
<td>TMR16B1PR</td>
<td>R/W</td>
<td>0x00C</td>
<td>Prescale Register (PR). When the Prescale Counter (below) is equal to this value, the next clock increments the TC and clears the PC.</td>
<td>0</td>
</tr>
<tr>
<td>TMR16B1PC</td>
<td>R/W</td>
<td>0x010</td>
<td>Prescale Counter (PC). The 16-bit PC is a counter which is incremented to the value stored in PR. When the value in PR is reached, the TC is incremented and the PC is cleared. The PC is observable and controllable through the bus interface.</td>
<td>0</td>
</tr>
<tr>
<td>TMR16B1MCR</td>
<td>R/W</td>
<td>0x014</td>
<td>Match Control Register (MCR). The MCR is used to control if an interrupt is generated and if the TC is reset when a Match occurs.</td>
<td>0</td>
</tr>
<tr>
<td>TMR16B1MR0</td>
<td>R/W</td>
<td>0x018</td>
<td>Match Register 0 (MR0). MR0 can be enabled through the MCR to reset the TC, stop both the TC and PC, and/or generate an interrupt every time MR0 matches the TC.</td>
<td>0</td>
</tr>
<tr>
<td>TMR16B1MR1</td>
<td>R/W</td>
<td>0x01C</td>
<td>Match Register 1 (MR1). See MR0 description.</td>
<td>0</td>
</tr>
<tr>
<td>TMR16B1MR2</td>
<td>R/W</td>
<td>0x020</td>
<td>Match Register 2 (MR2). See MR0 description.</td>
<td>0</td>
</tr>
<tr>
<td>TMR16B1MR3</td>
<td>R/W</td>
<td>0x024</td>
<td>Match Register 3 (MR3). See MR0 description.</td>
<td>0</td>
</tr>
<tr>
<td>TMR16B1CCR</td>
<td>R/W</td>
<td>0x028</td>
<td>Capture Control Register (CCR). The CCR controls which edges of the capture inputs are used to load the Capture Registers and whether or not an interrupt is generated when a capture takes place.</td>
<td>0</td>
</tr>
<tr>
<td>TMR16B1CR0</td>
<td>RO</td>
<td>0x02C</td>
<td>Capture Register 0 (CR0). CR0 is loaded with the value of TC when there is an event on the CT16B1_CAP0 input.</td>
<td>0</td>
</tr>
<tr>
<td>TMR16B1EMR</td>
<td>R/W</td>
<td>0x03C</td>
<td>External Match Register (EMR). The EMR controls the match function and the external match pins CT16B1_MAT[1:0].</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x040 - 0x06C</td>
<td>reserved</td>
<td>-</td>
</tr>
<tr>
<td>TMR16B1CTCR</td>
<td>R/W</td>
<td>0x070</td>
<td>Count Control Register (CTCR). The CTCR selects between Timer and Counter mode, and in Counter mode selects the signal and edge(s) for counting.</td>
<td>0</td>
</tr>
<tr>
<td>TMR16B1PWM</td>
<td>R/W</td>
<td>0x074</td>
<td>PWM Control Register (PWMCON). The PWMCON enables PWM mode for the external match pins CT16B1_MAT[1:0].</td>
<td>0</td>
</tr>
</tbody>
</table>

[1] Reset value reflects the data stored in used bits only. It does not include reserved bits content.