

PowerPC Elapsed Time

Microsecond timing

Thu, Jul 15, 1999

The PowerPC architecture includes a 64-bit TB TimeBase counter (in all chip models except the 601) that increments in units of something like a bus clock. This note shows how to derive a microsecond counter from this built-in counter. Although the PowerPC is *not* normally programmed in assembly language, this note describes a solution in those terms to illustrate the minimum that the code must do, no matter what language is actually used.

The TB register is a 64-bit counter that is always activated, accessed as two 32-bit values TBU and TBL. In order to assure integrity of the resulting 64-bit value, care must be taken. From the PowerPC documentation we have the following code in simplified assembler mnemonics that illustrates the method.

```
loop: mftbu    rx    #load most significant half from TBU
      mftb     ry    #load least significant half from TBL
      mftbu    rz    #load from TBU again
      cmpw     rz,rx #see if 'old' = 'new'
      bne     loop  #repeat if two values read from TBU are unequal
```

Following the above code, the 64-bit value is in (rx, ry), where rx contains the most significant 32-bits and ry the least significant 32-bits.

In order to compute a time value in microsecond units, we must scale the value obtained above. For definiteness in the following discussion, assume that the counting rate is about 66 MHz. To get microseconds, we must divide by 66. If the 66 value is not exact, we may introduce a slight error. But more importantly, the divide instruction is comparatively slow on the PowerPC. A 32-bit divide instruction might take 19 cycles, whereas a 32-multiply may be done in 4 cycles. And one would have to divide twice, as double precision is required. And to get the needed remainder, one must subtract the first product obtained. We can avoid the speed penalty by multiplying by the reciprocal of 66. This leads to the use of scaled integer calculations, if we wish to avoid using, and therefore having to save, floating point registers.

Consider the use of scaling by 2^{32} . This means we must multiply the 64-bit value by $2^{32}/66$, then shift right 32 bits to effect a divide by 2^{32} . Note that if the value 66 is not an exact integer—it has a fractional part—we preserve precision by this method. Further, note that a shift right of 32 bits is performed by merely selecting which word contains the result.

Assume that we have the 64-bit value in registers rx and ry as described above. Assume that the constant scale = $2^{32}/66$ has been computed ahead of time once and for all, when we don't care about the time required, and it is in register rz. The job here is to multiply a 32-bit value of scale by the 64-bit value obtained from the TB register, then divide by 2^{32} . First multiply scale by the TBL and produce a double precision (64-bit) product. Then multiply scale by the TBU and accumulate the lower half of the result with the previous upper half of the product. Normally, this would produce a 96-bit result, which

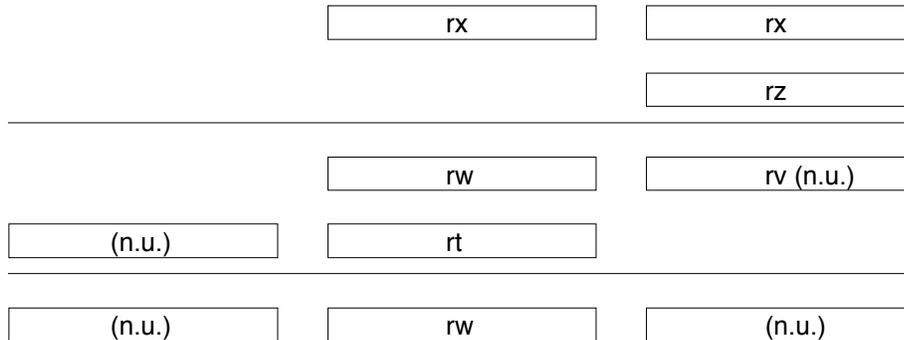
would then be "divided" by 2^{32} to produce a 64-bit end result in microsecond units. But since we only need a 32-bit answer, we just take the least significant half of this 64-bit end result. All of this analysis leads to a simplification of the steps that are required, as follows:

```

    mullw  rv,ry,rz #compute lower half of product of TBL*scale (not
needed!)
    mulhwu rw,ry,rz #compute upper half of product
    mullw  rt,rx,rz #compute lower half of product of TBH*scale
    add    rw,rw,rt #sum upper half of 1st product with lower half of 2nd

```

At this point, the result desired is in rw. Note that the first multiply is not needed and may be omitted. After obtaining the value of TB in two registers, and assuming the value of scale is in a third register, then only two multiplies and an add gives the resultant 32-bit time in microsecond units. To illustrate this, see as follows:



The diagram shows multiple precision multiplication analogous to the way we learned arithmetic in the early grades. Here, however, each box represents a 32-bit value, rather than a single decimal digit.

Another diagnostic time in use by the IRM-like system code is in units of 2000 Hz, or half-milliseconds. To do this case, the value for scale merely needs to be 500 times smaller, or $scale = 2^{32}/66/500$. The steps are the same as in the above logic, with the addition of complementing the result and returning the low byte, since the byte-wide counter was historically a decrementing counter.