6500 DATA WINDOWING EXAMPLES Application Note

Hi/fn® supplies two of the Internet's most important raw materials: compression and encryption. Hi/fn is also the world's first company to put both on a single chip, creating a processor that performs compression and encryption at a faster speed than a conventional CPU alone could handle, and for much less than the cost of a Pentium or comparable processor.

Hi/fn, Inc.
750 University Avenue
Los Gatos, CA 95032
info@hifn.com
http://www.hifn.com
Tel: 408-399-3500
Fax: 408-399-3501

Hi/fn Applications Support Hotline:
408-399-3544

Table of Contents

THIS PAGE INTENTIONALLY BLANK

1 Overview

The examples in this Application Note explain in detail how the byte ordering and word ordering windows of the 6500 Public Key Processor are used in specific instances. The register windows of the 6500 are actually very simple, requiring a very small amount of gates on the chip, but at first it may be bit difficult to understand intuitively the effect of (and even the need for) these transformations. In general, for any given system, it is likely that only a single register window needs to be used. It is hoped that the detailed real-life examples given here will help make it clear which window(s) should be used for a particular system.

Internally to the 6500 engines, no windowing bits are used or needed when accessing the data registers. The two windowing bits only affect transfers between the internal data registers and the outside world (e.g., the host CPU). The windowing bits do not affect CPU transfers to/from 6500 registers other than the data registers.

Bit 13 of the register address controls whether bits 2..6 of the register address presented to the chip are logically inverted when presented to the internal 6500 register array. This transformation has the effect of reversing the order of the 32-bit words within a register from the perspective of the outside world. Inside the 6500, this windowing bit 13 merely is used to XOR address bits 2..6 when the address accesses a data register.

Bit 14 of the register address controls whether the 32-bit word transferred over the CPU bus has a byte reversal (endian conversion) applied to it. The 6500 uses little-endian byte ordering internally, so this windowing bit allows automatic (and dynamic) endian conversion if necessary.

To keep the examples tractable, the integer values used below are smaller (e.g., 256 bits) than public key arithmetic values of practical interest (e.g., 1024 bits), but the same principles illustrated here can be extended to larger values in obvious ways.

In all cases, it is assumed in the discussions below that data bus bit 0 of the 6500 is connected to the least significant bit of the CPU data bus, and that data bus bit 31 of the 6500 is connected to the most significant bit of the CPU data bus.

Example:

Suppose that we wish to transfer a 256-bit integer value

```
N = 0x8F8E8D8C8B8A898887868584838281800001020304050607080 90A0B0C0D0E0F
```

into or out of the 6500 register space. The syntax used to represent this value is C-like;the most significant 8 bits of N are 0x8F, and the least significant byte of N is 0x0F.

Internal to the 6500, left-aligned little-endian ordering is used for modular arithmetic. Thus, if the register r3 (offset 0x180) holds N, the internal registers are:

```
internal register offset 0x01E0 = 0x0C0D0E0F    // least significant 32-bits
internal register offset 0x01E4 = 0x08090A0B
internal register offset 0x01E8 = 0x04050607
internal register offset 0x01EC = 0x00010203
internal register offset 0x01F0 = 0x83828180
internal register offset 0x01F4 = 0x87868584
internal register offset 0x01F8 = 0x8B8A8988
internal register offset 0x01FC = 0x8F8E8D8C    // most significant 32-bits
```

Let us now see how this value may be represented in the CPU main memory for different types of CPUs and various data value formats.

2 Little-endian CPU, little-endian data in memory

An example would be an Intel family CPU (e.g., Pentium) where data is kept in the 'natural' CPU order. At the byte level, the value of N in memory would be, in order of increasing memory address, starting at address X:

```
X     : 0x0F 0x0E 0x0D 0x0C 0x0B 0x0A 0x09 0x08
X+8   : 0x07 0x06 0x05 0x04 0x03 0x02 0x01 0x00
X+16  : 0x80 0x81 0x82 0x83 0x84 0x85 0x86 0x87
X+24  : 0x88 0x89 0x8A 0x8B 0x8C 0x8D 0x8E 0x8F
```

At the 32-bit word level, these bytes correspond to

```
X         : 0x0C0D0E0F
X+4       : 0x08090A0B
X+8       : 0x04050607
X+12      : 0x00010203
X+16      : 0x83828180
X+20      : 0x87868584
X+24      : 0x8B8A8988
X:28      : 0x8F8E8D8C
```

Note that, given this byte and word ordering in memory, the CPU can simply perform a memory move operation of length 32 bytes from address X to register offset 0x1E0 to register r3 in window 2. In 32-bit Intel assembly language, the data could be moved from memory into the 6500 as follows:

```
mov      edi,ADDR_6500_BASE   ; base address of the 6500
mov      eax,256              ; bit length
mov      [edi+100Ch],eax      ; set the length tag for r3
add      edi,41E0h            ; point to register r3, window 2
                              ; (left justify the 256-bit value)
cld                           ; incrementing addresses
lea      esi,X                ; start moving dwords from X
mov      ecx,256/32           ; move 8 dwords (256 bits)
rep movsd                     ; move the value from memory to r3
```

In other words, the sequence of operations is:

```
Memory read          6500 write      Internal register write
-------------------------------------------------------------
X     : 0x0C0D0E0F   → 0x41E0        0x0C0D0E0F   → 0x01E0
X+4   : 0x08090A0B   → 0x41E4        0x08090A0B   → 0x01E4
X+8   : 0x04050607   → 0x41E8        0x04050607   → 0x01E8
```

```
X+12  : 0x00010203   → 0x41EC        0x00010203   → 0x01EC
X+16  : 0x83828180   → 0x41F0        0x83828180   → 0x01F0
X+20  : 0x87868584   → 0x41F4        0x87868584   → 0x01F4
X+24  : 0x8B8A8988   → 0x41F8        0x8B8A8988   → 0x01F8
X:28  : 0x8F8E8D8C   → 0x41FC        0x8F8E8D8C   → 0x01FC
```

3 Little-endian CPU, big-endian data in memory

In many network protocols (e.g, IPSEC), the network byte ordering for public-key integer quantities is big-endian. In other words, the most significant byte of integer values used for RSA or Diffie-Hellman computations is sent over the line first, with the least significant byte last. Thus, even for a little-endian CPU such as the Intel Pentium, the data is stored in memory in big-endian order. When the public-key computations are performed in software, usually the first step is to convert from big-endian to little-endian. However, the 6500 windowing scheme allows simple transfers without any reordering in memory, as we shall see below.

At the byte level, the value of N in memory would be, in order of increasing memory address, starting at address Y:

```
Y       : 0x8F 0x8E 0x8D 0x8C 0x8B 0x8A 0x89 0x88
Y+8     : 0x87 0x86 0x85 0x84 0x83 0x82 0x81 0x80
Y+16    : 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07
Y+24    : 0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x0E 0x0F
```

At the 32-bit word level, these bytes appear to the CPU as

```
Y       : 0x8C8D8E8F
Y+4     : 0x88898A8B
Y+8     : 0x84858687
Y+12    : 0x80818283
Y+16    : 0x03020100
Y+20    : 0x07060504
Y+24    : 0x0B0A0908
Y+28    : 0x0F0E0D0C
```

Note that these 32-bit word values would be incorrect if the CPU tried to interpret them directly. However, given this byte and word ordering in memory, the little-endian CPU can simply perform a memory move operation of length 32 bytes from address Y to register offset 0x180 to register r3 in window 1, without any software transformations required. Because window 1 reverses the order of the word registers, the first 32-bit write to offset 0x6180 actually goes to internal data register address 0x1FC, which is the most significant word of the register. Because window 1 reverses the byte ordering, the conversion from big-endian to little-endian is also performed automatically by the 6500.

In 32-bit Intel assembly language, the data could be moved from memory into the 6500 as follows:

```
mov     edi,ADDR_6500_BASE   ; base address of the 6500
mov     eax,256              ; bit length
mov     [edi+100Ch],eax      ; set the length tag for r3
add     edi,2180h            ; point to register r3, window 3
cld                          ; incrementing addresses
lea     esi,Y                ; start moving dwords from Y
```

```
mov     ecx,256/32          ; move 8 dwords (256 bits)
rep movsd                   ; move the value from memory to r3
```

In other words, the sequence of operations is:

```
Memory read          6500 write      Internal register write
  ------------------------------------------------------------
  Y    : 0x8C8D8E8F    → 0x2180      0x8F8E8D8C     → 0x01FC
  Y+4  : 0x88898A8B    → 0x2184      0x8B8A8988     → 0x01F8
  Y+8  : 0x84858687    → 0x2188      0x87868584     → 0x01F4
  Y+12 : 0x80818283    → 0x218C      0x83828180     → 0x01F0
  Y+16 : 0x03020100    → 0x2190      0x00010203     → 0x01EC
  Y+20 : 0x07060504    → 0x2194      0x04050607     → 0x01E8
  Y+24 : 0x0B0A0908    → 0x2198      0x08090A0B     → 0x01E4
  Y+28 : 0x0F0E0D0C    → 0x219C      0x0C0D0E0F     → 0x01E0
```

4 Big-endian CPU, big-endian data in memory

On a big-endian CPU, such as the Motorola 68000 family, the IPSEC byte ordering is very natural for software public key computations. The 6500 windowing allows very simple transfers between CPU memory and the 6500 data registers.

At the byte level, the value of N in memory would be, in order of increasing memory address, starting at address Z:

```
Z       : 0x8F 0x8E 0x8D 0x8C 0x8B 0x8A 0x89 0x88
Z+8     : 0x87 0x86 0x85 0x84 0x83 0x82 0x81 0x80
Z+16    : 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07
Z+24    : 0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x0E 0x0F
```

At the 32-bit word level, these bytes appear to the CPU as

```
Z       : 0x8F8E8D8C
Z+4     : 0x8B8A8988
Z+8     : 0x87868584
Z+12    : 0x83828180
Z+16    : 0x00010203
Z+20    : 0x04050607
Z+24    : 0x08090A0B
Z+28    : 0x0C0D0E0F
```

Given this byte and word ordering in memory, the little-endian CPU can simply perform a memory move operation of length 32 bytes from address Y to register offset 0x180 to register r3 in window 3, without any software transformations required. Because window 3 reverses the order of the word registers, the first 32-bit write to offset 0x6180 actually goes to internal data register address 0x1FC, which is the most significant word of the register. Note that no endian conversion is required, since it is assumed that the CPU is wired to the 6500 data bus so that the most (and least) significant data bits of the two chips are connected.

In 68000 assembly language, the data could be moved from memory into the 6500 as follows:

```
        lea     (ADDR_6500_BASE),a0  ; base address of the 6500
        mov.L   #256,d0              ; bit length
        mov.L   d0,100Ch(a0)         ; set the length tag for r3
        add.L   #6180h,a0            ; point to register r3, window 1
        mov.L   #(256/32)-1,d1       ; move 8 longwords (256 bits)
        lea     (Z),a1               ; start moving longwords from Z
movLp:  mov.L    (a1)++,(a0)++       ; move one longword
        dbra    d1,movLp             ; loop until done
```

In other words, the sequence of operations is:

```
Memory read          6500 write      Internal register write
-------------------------------------------------------------
Z    :  0x8F8E8D8C   → 0x6180     0x8F8E8D8C      → 0x01FC
Z+4  :  0x8B8A8988   → 0x6184     0x8B8A8988      → 0x01F8
Z+8  :  0x87868584   → 0x6188     0x87868584      → 0x01F4
Z+12 :  0x83828180   → 0x618C     0x83828180      → 0x01F0
Z+16 :  0x00010203   → 0x6190     0x00010203      → 0x01EC
Z+20 :  0x04050607   → 0x6194     0x04050607      → 0x01E8
Z+24 :  0x08090A0B   → 0x6198     0x08090A0B      → 0x01E4
Z+28 :  0x0C0D0E0F   → 0x619C     0x0C0D0E0F      → 0x01E0
```

5 Big-endian CPU, little-endian data in memory

This scenario could occur with a big-endian CPU using a protocol with little-endian network ordering of public-key quantities. Since no such protocol of any importance is known, this case is left as an exercise for the reader. Window 1 of the 6500 would be used to re-order both bytes and words.