H Address Formation Details

This appendix provides detailed descriptions and illustrations of how various types of addresses are formed in a PA-RISC processor.

Memory Reference Instruction Address Formation

Addresses are formed by the combination of a Space ID and an address Offset. Address Offsets may be formed as the sum of a base register and any one of the following: a long displacement, a short displacement (which leaves more instruction bits for other functions), or an index register.

Long Displacement Addressing

Memory reference instruction formats that have long displacements form the effective memory reference address by adding a displacement to a base value specified through the instruction. The entity being transferred can be a doubleword, word, halfword, or a byte.

The displacement can be any of the following:

- a 16-bit byte displacement (restricted to 14 bits when PSW W-bit =0)
- a 12-bit word displacement for word loads and stores
- an 11-bit doubleword displacement for doubleword loads and stores.

The opcode specifies the particular data transfer to be performed and the form of the displacement. The displacements are encoded in two's complement notation with the sign bit always placed in instruction bit 31. The formats for long displacement instructions are:

<table>
<thead>
<tr>
<th>op</th>
<th>b</th>
<th>t/r</th>
<th>s</th>
<th>im14</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>op</th>
<th>b</th>
<th>t/r</th>
<th>s</th>
<th>im11a</th>
<th>op</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>11</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>op</th>
<th>b</th>
<th>t/r</th>
<th>s</th>
<th>im10a</th>
<th>m</th>
<th>op</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>10</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Space selection is done differently for 64-bit programs (when the PSW W-bit is 1) than it is for 32-bit programs (when the PSW W-bit is 0). For 64-bit programs, there is little need for providing direct user program access to an address space larger than 64 bits. Therefore, it is anticipated that most programs (other than system software) will use only implicit pointers. For this reason, for 64-bit programs, all long displacement loads and stores inherently compute their addresses as implicit pointers, and the s-field is used to extend the displacement by 2 bits, providing an effective 16-bit displacement. When the s-field is used to encode more displacement bits, they are encoded in a special fashion in order to allow...
the encoding for displacements which do not require the additional 2 bits of range to be the same for 32-bit and for 64-bit programs. (See the function “assemble_16” in “Instruction Notation Control Structures” on page E-1 for more details on encoding.) For 32-bit programs, the s-field simply specifies the space register.

When data translation is enabled, the effective space ID is the contents of a selected space register. If the PSW W-bit is 0 (a 32-bit program), and the s-field is non-zero, the space register is selected directly by the s-field (explicit pointer). If the PSW W-bit is 0 and the s-field is 0, the effective space ID is the contents of the space register whose number is the sum of 4 plus bits 32..33 of GR b (implicit 32-bit pointer). If the PSW W-bit is 1 (a 64-bit program), the effective space ID is the contents of the space register whose number is the sum of 4 plus bits 0..1 of GR b (implicit 64-bit pointer). When data translation is disabled, no space register selection is done and the offset is used directly as the address.

The effective offset is the sum of the contents of GR b and the sign-extended displacement d. For 32-bit programs, the offset is truncated to 32 bits (the upper 32 bits are forced to 0).

The address calculation is shown in Figure H-1 and Figure H-2 in three parts: Figure H-1 shows space identifier selection, and Figure H-2 shows offset computation. Space and offset are then bit-wise ORed, as shown in Figure H-3 to form the full virtual address.

![Figure H-1. Space Identifier Selection](image-url)
Base register modification can be optionally performed, and can either be done before or after the offset calculation, as shown in Figure H-2. Base register modification is specified by the opcode, or in the case of doubleword loads and stores, by the m-field.

**Short Displacement Addressing**

This section describes memory reference instruction formats, where the effective memory reference address is formed by the addition of a short 5-bit displacement to a base value specified in the instruction. The sign bit of the short displacement is the rightmost bit of the 5-bit field, which is in two’s complement notation. The entity being transferred can be a doubleword, word, halfword, or a byte.
The format of the short displacement load instructions is:

```
03  b  im5  s  a  1  cc  ext4  m  t
  6  5  5  2  1  1  2  4  1  5
```

and that of the short displacement stores is:

```
03  b  r  s  a  1  cc  ext4  m  im5
  6  5  5  2  1  1  2  4  1  5
```

The `ext4` field in the instruction format above specifies a load or a store and the data size. The `a` and `m` fields specify the following functions:

- \( a = 0 \) modify after if \( m = 1 \).
- \( a = 1 \) modify before if \( m = 1 \).
- \( m = 0 \) no address modification.
- \( m = 1 \) address modification.

In addition the combination \( a = 0, m = 1, \) and \( im5 = 0 \) specifies an ordered load or store.

The `cc` field specifies the cache control hint (see Table 6-7 on page 6-10, Table 6-8 on page 6-10, and Table 6-9 on page 6-11).

In the instruction descriptions that follow, some information is coded into the instruction names and the remainder is coded in the completer field (denoted by `cmplt` in the descriptions). Table H-1 lists the assembly language syntax of the completer, the functions performed, and the values coded into the `a`, `m`, and `im5` bit fields of the instruction.

<table>
<thead>
<tr>
<th>cmplt</th>
<th>Description</th>
<th>a</th>
<th>m</th>
<th>im5</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;none&gt;</td>
<td>don’t modify base register</td>
<td>x</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>MA</td>
<td>Modify base register After</td>
<td>0</td>
<td>1</td>
<td>≠0</td>
</tr>
<tr>
<td>MB</td>
<td>Modify base register Before</td>
<td>1</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>O</td>
<td>Ordered access</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Notes: x indicates don’t care.

In the above table, `cmplt` is in assembly language format and `a`, `m`, and `im5` are in machine language format.

The space identifier is computed like any other data memory reference (see Figure H-1 on page H-2). The calculation of the offset portion of the effective address for different completers is shown in Figure H-4. Space and offset are combined like any other data memory reference (see Figure H-3 on page H-3).
Offset Computation
,MB Completer

Offset Computation
,MA Completer

Offset Computation
No Completer Specified

Figure H-4. Offset computation with short displacement
Store Bytes Instructions

STORE BYTES and STORE DOUBLEWORD BYTES provide the means for doing unaligned byte moves efficiently. These instructions use a short 5-bit displacement to store bytes to unaligned destinations. The short displacement field is in two’s complement notation with the sign bit as its rightmost bit.

The format of the STORE BYTES and STORE DOUBLEWORD BYTES instructions is:

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
\text{03} & b & r & s & a & 1 & \text{cc} & \text{ext4} & m & \text{im5} \\
\hline
6 & 5 & 5 & 2 & 1 & 1 & 2 & 4 & 1 & 5 \\
\end{array}
\]

The \textit{ext4} field in the instruction format above specifies the data size. The \textit{a} and \textit{m} fields specify the following functions:

\[
\begin{align*}
 \text{a} = 0 & \quad \text{store bytes beginning at the effective byte address in the word or doubleword.} \\
\text{a} = 1 & \quad \text{store bytes ending at the effective byte address in the word or doubleword.} \\
\text{m} = 0 & \quad \text{no address modification.} \\
\text{m} = 1 & \quad \text{address modification.}
\end{align*}
\]

The \textit{cc} field specifies the cache control hint (see Table H-2 on page H-6).

In the instruction descriptions that follow, some information is coded into the instruction names and the remainder is coded in the completer field (denoted by \textit{cmplt} in the descriptions). Table H-2 lists the assembly language syntax of the completer, the functions performed, and the values coded into the \textit{a} and \textit{m} fields of the instruction.

<table>
<thead>
<tr>
<th>\textit{cmplt}</th>
<th>Description</th>
<th>\textit{a}</th>
<th>\textit{m}</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;none&gt; or B</td>
<td>Beginning case, don’t modify base register</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B,M</td>
<td>Beginning case, Modify base register</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>Ending case, don’t modify base register</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>E,M</td>
<td>Ending case, Modify base register</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

In the above table, \textit{cmplt} is in assembly language format and \textit{a} and \textit{m} are in machine language format.

The space identifier is computed like any other data memory reference (see Figure H-1 on page H-2). The calculation of the offset portion of the effective address for different completers is shown in Figure H-5. Space and offset are combined like any other data memory reference (see Figure H-3 on page H-3).

The actual offset and modified address involves some alignment and other considerations. Refer to the instruction description pages for an exact definition.
Figure H-5. Offset computation for Store Bytes and Store Doubleword Bytes
Indexed Addressing

This section describes memory reference instruction formats, where the effective memory reference address is formed by the addition of an index value to a base value specified in the instruction. The entity being transferred can be a doubleword, word, halfword, or a byte.

The format for indexed instructions is:

<table>
<thead>
<tr>
<th>cmplt</th>
<th>Description</th>
<th>u</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;none&gt;</td>
<td>no index shift, don’t modify base register</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>M</td>
<td>no index shift, Modify base register</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td>Shift index by data size, don’t modify base register</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SM or S,M</td>
<td>Shift index by data size, Modify base register</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

In the above table, *cmplt* is in assembly language format and *u* and *m* are in machine language format.

The space identifier is computed like any other data memory reference (see Figure H-1 on page H-2). The calculation of the offset portion of the effective address for different completers is shown in Figure H-6. Space and offset are combined like any other data memory reference (see Figure H-3 on page H-3).
Absolute Address Formation

The formation of absolute addresses varies depending on the setting of the PSW W-bit which specifies whether the system is to support full 64-bit offsets or the 32-bit offsets compatible with PA-RISC 1.0 and 1.1 systems.
Absolute Accesses when PSW W-bit is 1

When the PSW W-bit is 1 (see “Processor Status Word (PSW)” on page 2-7 for the definition of the PSW W-bit), an absolute address is a 62-bit unsigned integer whose value is the address of the lowest-addressed byte of the operand it designates (see Figure H-7).

<table>
<thead>
<tr>
<th>ne</th>
<th>Absolute Byte Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>62</td>
</tr>
</tbody>
</table>

Figure H-7. 62-bit Absolute Pointer

The 2-bit $ne$ field is a non-existent field (i.e., software may write any value, but hardware implementations must ignore them).

A 64-bit physical address is formed by extending a 62-bit absolute address as shown in Figure H-8 and described by the following pseudo-code:

```plaintext
if (abs_addr{2..9} != 0xF0) { /* if not in PDC Address Space */
    phys_addr{2..63} ← abs_addr{2..63};
    if (abs_addr{2..5} == 0xF) /* if I/O Address Space */
        phys_addr{0..1} ← 0x3;
    else /* if Memory Address Space */
        phys_addr{0..1} ← 0x0;
} else { /* if PDC Address Space */
    phys_addr{0..7} ← 0xF0;
    phys_addr{8..9} ← processor-specific;
    phys_addr{10..63} ← abs_addr{10..63};
}
```

**NOTE**

Restricting absolute addresses when the PSW W-bit is 1 to 62 bits in size enables software to access any objects in a quarter of the 64-bit Physical Address Space by two means:

- Using a 62-bit absolute address
- Using a virtual address which implicitly uses any of Space Registers 4 through 7.

Maintaining a 64-bit virtual address space enables software to virtually access hardware subsystems such as I/O busses which define 64 bit physical addresses.
Absolute Accesses when PSW W-bit is 0

When the PSW W-bit is 0, an absolute address is a 32-bit unsigned integer whose value is the address of the lowest-addressed byte of the operand it designates (see Figure H-9)

<table>
<thead>
<tr>
<th>non-existent</th>
<th>Absolute Byte Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>32</td>
</tr>
</tbody>
</table>

Figure H-9. 32-bit Absolute Pointer

A 64-bit physical address is formed by extending the 32-bit Absolute Address as shown in Figure H-10 and described by the following pseudo-code:
if (abs_addr{32..39} != 0xF0) {
    phys_addr{32..63} ← abs_addr{32..63};
    if (abs_addr{32..35} == 0xF)
        phys_addr{0..31} ← 0xFFFFFFFF;
    else
        phys_addr{0..31} ← 0x00000000;
} else {
    phys_addr{0..7} ← 0xF0;
    phys_addr{8..39} ← processor-specific;
    phys_addr{40..63} ← abs_addr{40..63};
}

/* if not in PDC Address Space */
/* if I/O Address Space */
/* if Memory Address Space */
/* if PDC Address Space */

Figure H-10. 32-bit Absolute Accesses when PSW W-bit is 0
Figure H-11 illustrates the relationship between the 64-bit Physical Address Space, absolute accesses when the PSW W-bit is 0, and an example, 40-bit, implemented physical address space.

Figure H-11. Physical Address Space Mapping - An Example