Op Code	R_1 5	I_2I_2	I_2I_2	I_2I_2	I_2I_2

The Branch Relative and Save Long instruction functions in a manner that is similar to BAS in that it saves the return address and branches to a target location denoted by operand 2. The mechanism for branching is different however. The four-byte immediate constant is treated as a 32-bit two's complement integer which is doubled to represent the number of halfwords from the current instruction to the target address. The doubled integer value is added to the PSW instruction address to determine the effective target address. No base registers are involved. Since the integer value can be positive or negative, forward and backward branching is possible. With 32 bits to represent a number of halfwords, the target address is plus or minus 64 GiB away from the BRASL instruction.

In 24-bit or 31-bit addressing mode, some linkage information - the basic addressing mode bit 32 (0 -24 or 31 bits, 1 – 64 bits) as well as the rightmost 31 bits of the instruction address (bits 97-127 of the PSW) - is placed in bit positions 32 and 33-63, respectively, of the first operand register. Bits 0-31 of operand 1 remain unchanged. In 64-bit addressing mode, linkage information consisting of the updated instruction address, is placed in bit positions 0-63 of operand 1. The condition code is unaffected by this operation.

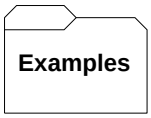
When BRASL is the target of an EX instruction, the branch is relative to the location of the target BRASL.

BRASL offers significant advantages over BAS. BAS requires a base register to cover the target address, but BRASL uses a jump mechanism from the current instruction which frees us from having to use a base register for addressability to the target. Using this instruction, and other similar relative branch instructions, it is possible to write long blocks of code that don't require base registers. Relative branching instructions are some of the most important instructions available on Z-class machines.

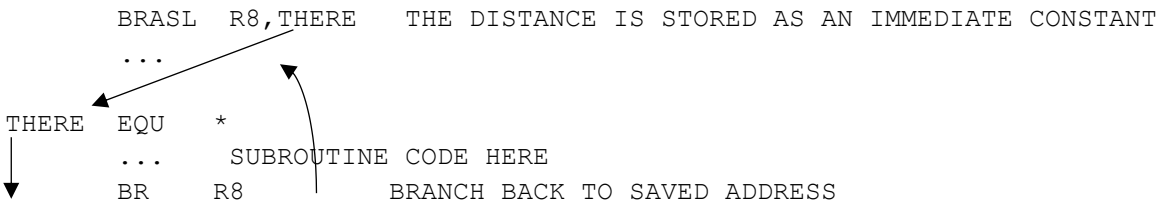
Tips

1) Choose BRASL over BAS. Choose BRASL over BRAS only if BRAS cannot generate the required relative constant. There's no real drawback to using relative branches, and you may be able to save a base register required for based branches. With relative branches, the target address is

specified as a number of halfwords from the current instruction. No base registers are needed for target addresses.



A Example Branch Relative and Save



Trying It Out in VisibleZ:

1) Load the program **brasl.obj** from the \Codes directory and cycle through the first instruction which establishes R12 with a base address. The second instruction is L, which initializes 32 bits of R8 with 1's. What is the value of the relative immediate constant in the BRAS instruction? From what address the the relative branch forward measured?

2) Load the program **brasl1.obj** and cycle through the first three BRC instructions. Why does this program go into an infinite loop?

3) Load the program **brasl2.obj**. Why does this program cycle back to the beginning when it executes the BRAS instruction? What is the value of the relative immediate constant?