Overview of the Implementation of the PA 7100LC Multimedia Enhancements

One goal in adding the multimedia instructions was to minimize the amount of new circuits to be added to the existing ALUs and to minimize the impact on the rest of the CPU. This goal was accomplished. The only circuit changes to the CPU were in the ALU data path and decoder circuits. These instructions reuse most of the existing functionality and very small modifications and additions were required to implement them.

All of the new instructions implemented require two 16-bit adds or subtracts to be done in parallel. The existing ALU adder was modified to provide this functionality. These instructions required that the existing 32-bit adder be conditionally split into two 16-bit halves without sacrificing the performance of the 32-bit add. Conceptually this is equivalent to blocking the carry from bit 16 to bit 15 in a ripple-carry adder. To accomplish this, we made the following modifications.

The ALU adder is similar to a carry lookahead adder. The first stage of the adder calculates a carry generate and a carry propagate signal for each single bit in the adder. In this case, 32 single-bit generate and 32 single-bit propagate signals are calculated. These single-bit carry generate and carry propagate signals are used in subsequent stages of the carry chain to calculate carry generate and carry propagate signals for groups of bits.

The 32-bit adder was divided into two 16-bit halves between bits 15 and 16 by providing alternate signals for the carry generate and carry propagate signals from bit 16 (Fig. 1). The new generate and propagate signals from bit 16 are created with a two-input multiplexer. When a 32-bit addition or subtraction is being performed, the multiplexer selects the original generate and propagate signals to be passed onto the next stage of the carry chain. When 16-bit addition or subtraction is being performed the multiplexer selects the value for generate and propagate from the second input which is false (logical 0) for additions and true (logical 1) for subtractions.

The new generate and propagate signals can be forced to be false for instructions requiring halfword addition. This stops the carry from being generated by bit 16 or propagating from bit 16 to bit 15, even if this generate and propagate signal is not used directly to calculate the carry signal (as is the case in this adder). The generate and propagate signals can also be forced to be true for instructions requiring halfword subtraction. This will force a carry into the more significant halfword of the adder by generating a carry from bit 16 into bit 15. This technique is used along with the ones complement of the operand to be subtracted to perform subtraction as twos complement addition.

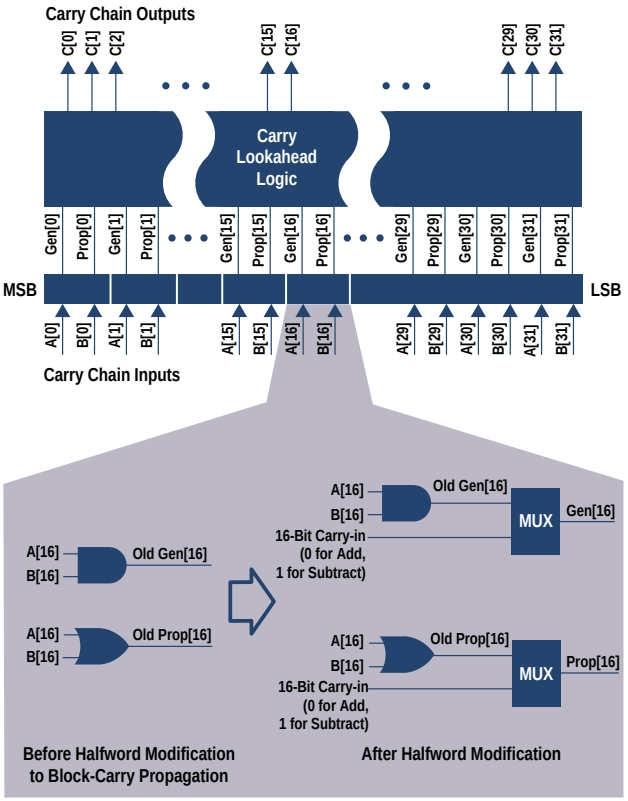

Fig. 1. Modifications to the carry lookahead adder to accommodate the halfword instructions.

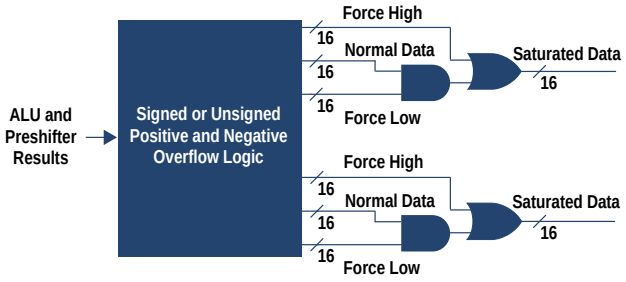

Fig. 2. Saturation logic. There is one of the these circuits for each halfword.

The original carry generate and propagate signals from bit 16 are still generated to calculate overflows from the less significant halfword addition. This overflow is used by the saturation logic, which can be invoked by some of these instructions.

Saturation requires groups of bits of the result to be forced to states of true or false, or passed unchanged. This is accomplished with an AND-OR gate (Fig. 2). The AND function can force the output of the gate to be false and the OR function can force the output of the gate to be true. Thus, the output is either forced high, forced low, or forced neither high nor low. It is never simultaneously forced high and low. The key is to determine when to force the result to a saturated value.

The saturation circuit is added at the end of the ALU's data path after the result selection multiplexer selects one of the results from the adder after it performs additions, subtractions, or logical operations such as bitwise AND, OR, or XOR (Fig. 3). The saturation circuit does not impact the critical speed paths of the ALU because it is downstream from the point where the cache data address is driven from the adder and where the test condition logic (i.e., logic for conditional branch instructions) obtains the results from which to calculate a test condition.

If signed saturation is selected, the ALU will force any 16-bit result that is larger than 0x7fff to 0x7fff (2^{15}–1) and any 16-bit result that is smaller than 0x8000 to 0x8000 (-2^{15}). These conditions represent positive and negative overflow of signed numbers. Positive and negative overflow can be detected by examining the sign bit (the MSB) of each operand and the result of the add. If both operands are positive and the result is negative then a positive overflow has occurred and the result in this case is saturated by forcing the most-significant bit to a logical 0 and the rest of the bits to a logical 1. If both operands are negative and the result is positive then a negative overflow has occurred and the result in this case is saturated by forcing the most significant bit to a logical 1 and the rest of the bits to a logical 0. Unsigned saturation is implemented in a similar way.

The average instruction, HAVE, requires manipulating the result after the addition is finished. Before the implementation of the halfword instructions the ALU selected between the results of a bitwise AND, a bitwise OR, a bitwise XOR, or the sum of the two input operands. The halfword average instruction adds an additional choice. The average result is the sum of the two input operands shifted right one bit position with a carry out of the most-significant bit (MSB) becoming the MSB of the result. To perform rounding of the result, the least-significant bit (LSB) of the result is replaced by an OR of the two least-significant bits before shifting right one bit.

The shift right and add and the shift left and add functions were added by modifying the x-bus preshifter in the operand selection logic of the ALU. The original ALU was capable of shifting 32-bit inputs left by zero, one, two, or three bits. To implement the 16-bit shift left and add instructions, the left-shift circuits had to be broken at

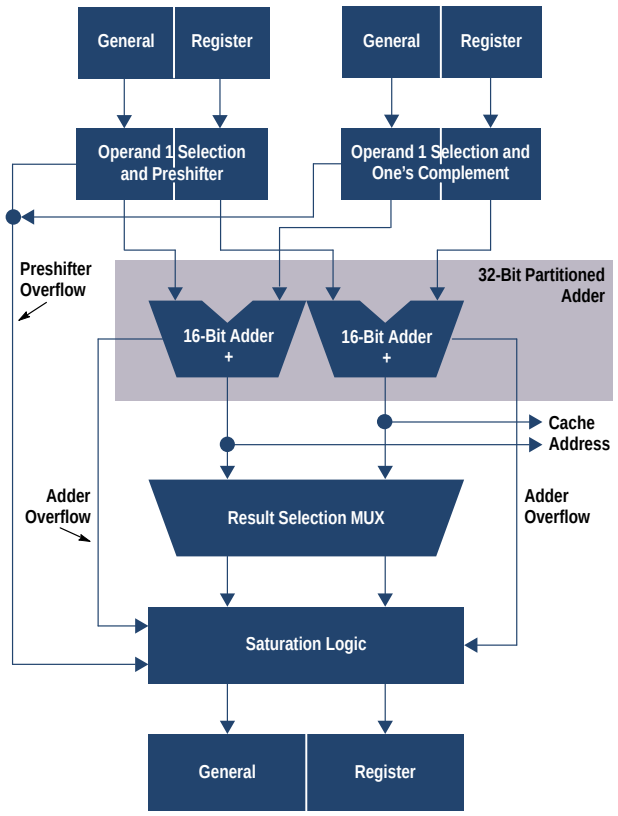

Fig. 3. Flow of halfword instructions showing the location of the saturation logic in relation to the ALU.

the halfword boundary. This was done by ANDing the bits shifted from the least-significant halfword to the most-significant halfword with a control signal that indicates when a 32-bit shift is being done. The 16-bit shift right and add instructions were implemented by adding the ability to shift one, two, or three bits right. This shift is always broken at the halfword boundary.

One challenging aspect of implementing the 16-bit shift left and add instructions was detecting when the results of shifting an operand left by one, two, or three bits causes a positive or negative overflow. A positive overflow occurs when the unshifted operand is positive and a logical one is shifted out of the left, or when the result of the shift is negative. A negative overflow occurs when the unshifted operand is negative and a logical zero bit is shifted out of the left, or when the result of the shift is positive. These overflow conditions are combined with the overflows calculated by the adder and used to saturate the final result. The final result is saturated if either the left shift or the adder causes an overflow.

The result of selecting instructions that can provide the most useful functionality while costing the least to implement was a relatively small increase in the area of the ALU. About 15% of the ALU's area is devoted to halfword instructions. Since the ALU's circuits were the only ones modified on the processor chip, only about 0.2% of the total processor's chip area is devoted to halfword instructions.