Unified Datapath: An Innovative Approach to the Design of a Low-Cost, Low-Power, High-Performance Microprocessor

Norman Yeung, Barbara Zivkov, Gulbin Ezer

MIPS Technologies, Inc., Silicon Graphics, Inc.
2011 N. Shoreline Blvd, Mountain View, CA, 94039

Abstract

Unifying the integer and floating-point datapath of the R4200 MIPS microprocessor allows for the sharing of some of the major hardware resources within the integer and floating-point execution units, and simplifies a large portion of the peripheral circuitry. The unified datapath results in a more efficient use of hardware with reduced average power dissipation, area and development time, without compromising the major performance advantages of RISC processors.

1: Introduction

The MIPS R4200 is a low-power low-cost high-performance 64-bit RISC microprocessor that is targeted at the cost- and power-sensitive notebook PC and embedded markets. To make this processor a successful product in its targeted markets, the objectives in the design of the R4200 were set to:

- Minimize power consumption (< 2watts)
- Minimize use of real estate (< 10mmx10mm)
- Minimize time to market (< 18 months)
- Provide high integer performance (> 50 iSPEC)
- Implement MIPS-II and MIPS-III ISA
- Comply with IEEE-754 with software support

To implement MIPS' instruction set architecture and to comply with IEEE-754, the processor's datapath must handle all four types of data formats:

- 32-bit single-precision floating-point
- 64-bit double-precision floating-point
- 32-bit word fixed-point
- 64-bit double-word fixed-point

In order to achieve the project objectives -- low power, low cost and instruction-level compatibility, while providing a high level of integer performance, the R4200 execution unit must maximize the usage of each and every one of its functional units.

After a close examination of the data types and the required instruction sets, we realized that the 64-bit integer datapath optimally overlaid the 53-bit mantissa datapath required for double-precision floating point operations. Since for the R4200's target applications, floating-point performance is less critical than integer performance, the floating-point mantissa datapath was merged with the integer datapath, resulting in this unified integer and floating-point datapath.

The following describes the functionalities of the unified datapath and its elements. Examples are presented to show how efficiently this datapath processes the integer and floating-point instructions. Finally, the advantages and disadvantages of this datapath design are examined.

2: Overview of the Unified Datapath

The R4200 instruction execution unit is tightly coupled to the on-chip cache memory system, I/DCache, and the on-chip memory management unit. The unit has a multifunction 5-stage pipe and is responsible for the execution of:

- Integer arithmetic and logic instructions
- Floating-point co-processor instructions
- Branch/jump instructions
- Load/store instructions

The execution unit datapath contains a 64-bit wide unified integer/mantissa datapath and a dedicated 12-bit wide exponent datapath.

The integer/mantissa datapath is compatible with both 32-bit and 64-bit operands for integer and floating-point numbers. Proper sign extension or zero-fill is performed on the operands prior to any computation. The exponent datapath is a 12-bit dual carry-select adder which is used for exponent subtraction, pre-

1063-6390/94 $3.00 © 1994 IEEE

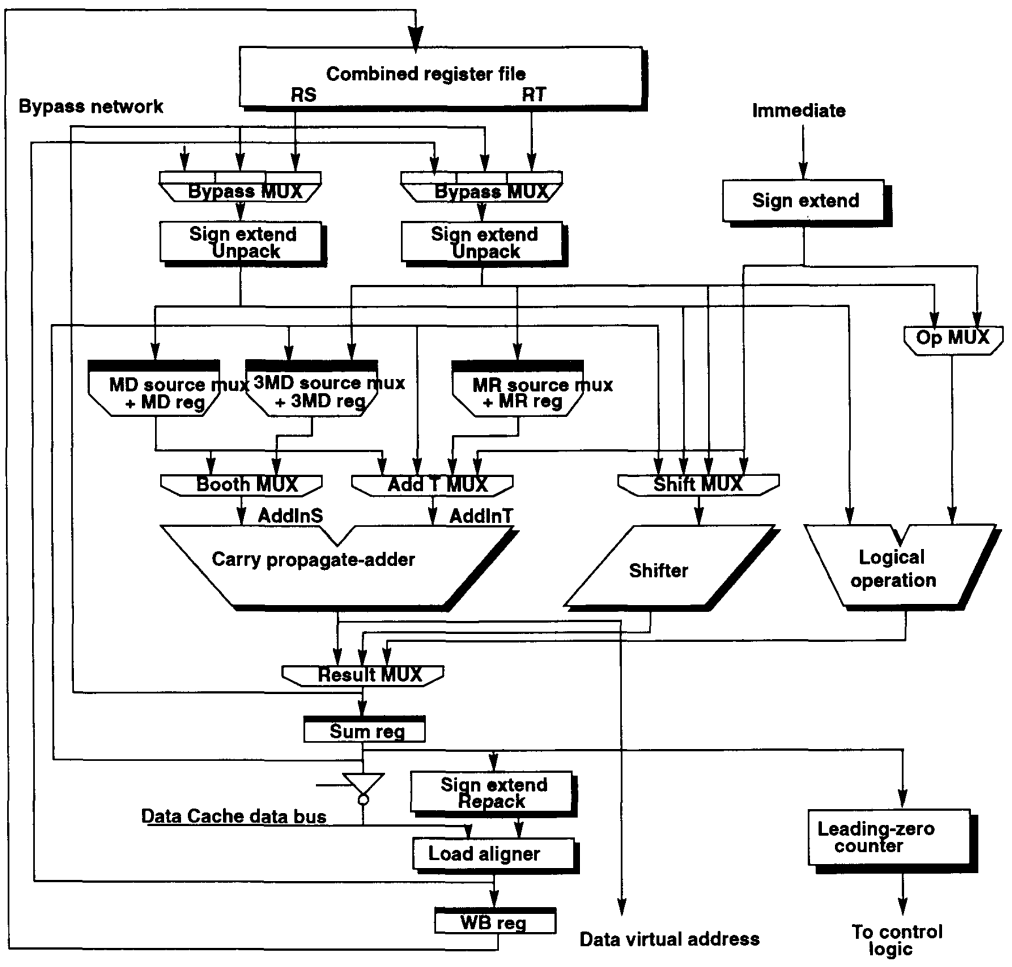

Figure 1: Integer/mantissa datapath

alignment shift calculation, and exponent addition for post-normalization final exponent update. Figure 1: Integer/mantissa datapath shows the functional units of the main execution datapath and Figure 2: Exponent datapath shows the functional units of the exponent datapath.

The integer/mantissa datapath consists of a register file, an unpacker, a repacker, an adder, a shifter, a leading-zero counter, a logical unit, a load aligner, and an operand bypass network. All of these functional units are used in both integer and floating-point instruction processing, except the leading-zero counter.

2.1: Register File

The execution unit can logically be thought of as having a general purpose register file consisting of thirty-two 64-bit integer registers and a floating-point register file consisting of thirty-two 64-bit floating-point registers. The R4200 implements a single sixty-four by 64-bit register file with two read ports and one write port. This implementation saves area by eliminating the requirement for dual address decoders and extra data busses for transferring data between the CPU and FPU.

To save power, each read port can be disabled independently for instructions not requiring a read operand

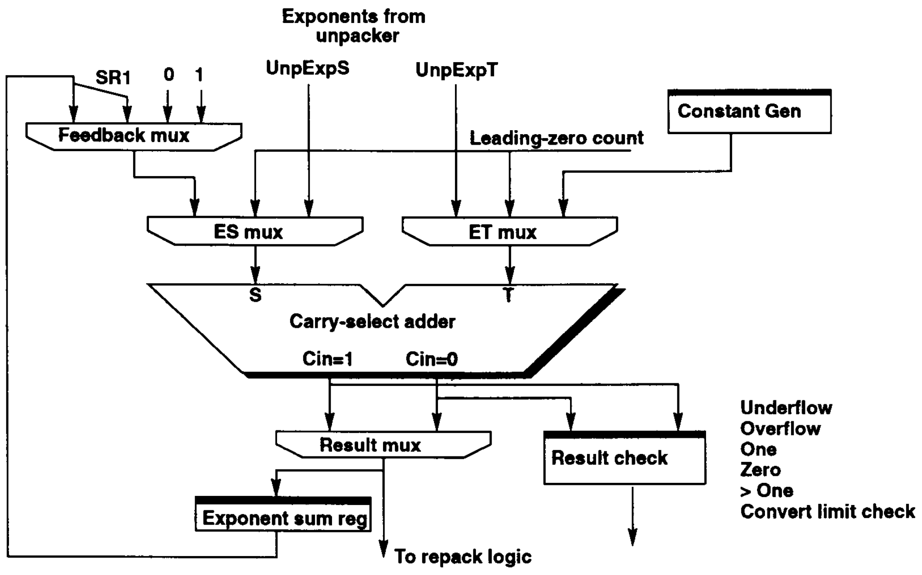

Figure 2: Exponent datapath

from that port or at certain times when the read operand is being bypassed from a later pipeline stage. To save power during stalls, the register file is only written during the first pipeline clock PCLK of the writeback WB stage. This prevents the rewriting of data to the register file should the pipeline stall with an instruction writing to the register file in the WB stage. Data is written to the register file during phase 1 of the WB stage and read during phase 2 of the RF stage, eliminating the need for extra data busses to bypass data from the WB stage to the RF since the new data can be read directly from the register file.

2.2: Operand Bypass

For the sake of performance, it is necessary for the results in two stages, the instruction execution EX stage and the data cache access DC stage, to be available to subsequent instructions waiting for it in the following EX stage as the source operands RS and/or RT. Thus, the operand bypass network is built into the datapath to allow feedback of results registered from the EX and DC stages to the input of EX stage.

2.3: Unpacker/Repacker

For all floating-point source operands, the unpacker breaks down single- and double-precision formatted operands into sign, exponent and mantissa fields, while the repacker performs the reverse operations.

As parts of the unpacker/repacker function, operands read from the register file are sign-extended by the unpacker if required for integer operations. The result is sign-extended if necessary for integer operations by the repacker before it is written back to the register file. Sign-extending the integer operations permits the use of single sign, carry out, and overflow bits from the functional units, independent of the format (word vs. double word) of an operation.

2.4: Adder

The integer/mantissa datapath has only one carry-select adder for all integer and floating-point computation instructions. The adder needs to be wide enough to accommodate minus four times the multiplicand (-4MD). Thus, the adder must be 67 bits wide -- 64 bits for an unsigned double word integer, plus two bits to be able to multiply it by 4, and a sign bit. Disregarding the input operands' data type, the adder always performs addition of two 67-bit operands in two's compliment form and a carry-in. The adder produces a 67 bit sum and a carry out.

The adder is also used to compute data virtual address for load and store, and to compare two operands in trap instructions.

2.5: Shifter

The unified datapath has one bidirectional shifter for all integer and floating-point computation instructions. The shifter has built-in guard/round/sticky collection logic for an FP pre-alignment shift and is also responsible for FP post-normalization, integer variable shift, and store alignment shift. The shifter is a 64-bit left/right shifter with a post word swap for store alignment. Right shifts may be either arithmetic, where the result is sign-extended, or logical, where the result is zero filled.

2.6: Leading Zero Counter

The leading zero counter is used in floating-point addition and subtraction operations to indicate how many places the sum or difference needs to be shifted to produce a normalized result, and to indicate how much the exponent needs to be decremented.

This counter is also used in convert operations for normalization and exponent calculation.

2.7: Logical Unit

The logical unit performs bit-wise boolean operations as well detecting equal operands for integer and floating-point operand comparisons and determining the outcome of conditional branches and traps.

2.8: Load Aligner

For load instructions, it is necessary to maintain a load delay of one PCLK for performance reason. Due to the timing requirements imposed by this load delay, a dedicated byte-wide shifter called load aligner is needed to shift the memory read data in bytes, halfwords, and words in the right or left direction.

2.9: Exponent Datapath

The exponent datapath is dedicated for floating-point data processing. It consists of a feedback mux and 2 operand muxes to select the inputs to the adder, constant-generating logic, a carry select adder, random logic to perform exception detection, and a register to hold the selected result from the adder.

3: Instruction Processing

Each functional unit in the datapath can be efficiently and effectively designed to handle various data types in distinctively different instruction sets for CPU and FPU. For example, a shifter can be optimized to handle three very different classes of instructions -- integer shift, floating-point add, and store instructions.

3.1: Integer Shift

For the class of integer shift instructions, the shifter performs both logical and arithmetic shift operations. It shifts either left or right on words or double-words by a fixed or variable amount. For shift right instructions, the shifter sign-extends the shifted data during an arithmetic shift, or zero-fills during a logical shift.

3.2: Floating-Point Add

There are four basic steps to perform a floating-point addition: (1) The exponents of the two operands are subtracted. The mantissa with the smaller exponent is pre-aligned with the other by right-shifting it by the difference of the exponents. (2) The mantissas are added. (3) The sum is rounded. (4) The rounded result is post-normalized by shifting it left until the leftmost "1" is in the integer bit position for the given format.

For floating-point addition, the shifter performs both the pre-alignment and post-normalization shift operations. For pre-alignment, the shifter performs a right shift by the amount determined by the difference of the exponents, and collects the guard, round and sticky bits.

For post-normalization, the shifter performs a left shift operation on the intermediate sum and its guard and sticky bits, by the amount calculated by the leading zero counter.

3.3: Store

For store instructions, the shifter performs data alignment operation on the register read data, depending on (1) the operand size -- byte, half-word, full-word, double-word, (2) the store operation type -- left or right, (3) the byte address and (4) processor byte order -- big- or little-endian mode.

3.4: Integer/Floating-Point Multiply

Not only can the functional units be efficiently designed to handle different instruction classes, the overall unified datapath can also efficiently support both integer and floating-point computations, with its shared resources. For example, an integer multiply uses the same resources in the unified datapath as that in a floating-point multiply.

The integer and floating-point multiplies are performed using a third-order (radix 8) modified booth recoding; retiring three multiplier bits per cycle. The partial product is produced using the datapath's carry propagate adder and thus is kept in non-redundant form.

Integer multiply and the mantissa portion of floating point multiply are almost identical from a functional point of view. In all multiplies, the first cycle is used to form the 3 times multiplicand required for 3rd order

Instruction	Pipeline clock cycles[1] Int(32)	Int(64)	FP(SP)	FP(DP)
Add/Sub	1	1	3	3
Multiply	13	24	11	20
Divide	39	71	29	58
Sqrt	-	-	31	60
Abs/Neg	-	-	1	1
Compare	-	-	1	1
DP->SP	-	-	-	2
SP->DP	-	-	1	-
FP->Int	-	-	5	5
Int->FP	-	-	5	5

Note:
1. The trivial cases for the multicycle FP instructions take two pipeline clocks to complete.

Table 1: Integer and Floating-point instruction latencies

booth recoding. In subsequent cycles 3 bits are shifted off the LSB end of the multiplier and are fed to booth logic which selects the appropriate multiple of the multiplicand to be added to the partial remainder. The only differences are: (1) the number of iterations required, (2) the 3 bits that are shifted off the right end of the partial remainder each cycle are accumulated in the MIPS architecturally defined register LO for integer multiplies and are ORed together to form a sticky bit for floating point, (3) the integer result is exact whereas the floating result is rounded, and (4) the integer result is stored in registers HI and LO whereas the floating-point result is returned to a register in the floating-point register file.

As mentioned above, the first cycle is used to form the 3-times multiplicand. This is achieved by loading the multiplicand (MD) into the MD source mux where it is then fed to the MD register as well as the MR mux which feeds it to the adder input AddInT, and the booth mux where the version of MD shifted left by one is fed to the adder input AddInS. The result of the add is fed to the sum register. During this first cycle, the multiplier is loaded into the MR register through the MR source mux.

On the second cycle, the 3-times multiplicand sum is selected by the 3MD source mux and the 3MD shift mux where it is fed to the 3MD register and is available to the booth mux. Since the 3-times multiplicand will be latched into the 3MD reg at the end of this cycle, it will be available for all subsequent cycles. The MR source mux shifts the value in the MR reg right by three for the floating point multiply on this cycle. This is done to pass over the guard, round and sticky bits which are all zero. The three LSBs coming out of the MR source mux are fed into the booth logic where they along with the MSB of the 3 bits from the previous cycle (zero for the first iteration) are recoded to select the first multiple of the multiplicand. Note that the booth recode is actually performed one cycle before it is used and the recoded selects are registered. The booth logic sends the select signals to the booth mux which feeds the AddInS of the adder. The MR mux selects zero on this first iteration and on subsequent iterations it selects the right shifted partial remainder, as the input to AddInT. The result of the adder is shifted right by 3, with the three LSBs right shifted into the LO register for integer operations (into bits 63:61 for double-word multiplies, bits 31:29 plus sign extension if required for word multiplies), or into an OR gate with the fourth LSB to form the current sticky bit for floating point operations. The shifted output of the adder is captured in the Sum register and the sticky bit becomes the new LSB.

The multiplier iterates by repeating the previous step (except 3MD is available from the 3MD register rather than SumEx) and stopped the cycle after there are less than 3 bits of multiplier left in the MR register.

Once all the iterations are complete, the execution EX stage of the integer multiply is complete. The output of the adder is latched, with sign extension if necessary, into HI.

The floating-point multiply, on the other hand, still requires rounding, exponent calculation and error checking before the EX stage is complete. Once all the multiplier bits have been retired, the rounding logic looks at the LSBs and the two MSB, along with the rounding mode, and decides whether a round up is called for. The product is fed into the Booth Mux and a "1" is forced into the Add T mux at the appropriate position determined by the round logic if round up is called for. The exponent is calculated by adding the biased exponents of the multiplier and the multiplicand and then subtracting the bias. If the rounded product is greater than or equal to two then the product is shifted right by 1 and exponent + 1 is selected to be returned to the repacker.

4: Conclusion

The 64-bit unified datapath in the R4200 combines the integer and floating-point datapath into a single datapath with shared resources. As demonstrated in the shifter example, many functional units required by integer operations can be shared by floating-point operations with minimal increase in their complexity.

Besides the obvious benefit of reducing the number of computational resources, the unified datapath completely eliminates the need for busses transferring data between CPU and FPU. Also, it makes bypassing data between the processing units more efficient by means of its single pipeline design.

Now that integer and floating-point instructions are processed by a single datapath, the control logic can be partitioned on the basis of functionality, not on the boundary between the on-chip processing units, CPU and FPU. For example, integer multiply instructions and floating-point multiply instructions are processed by a single multiplier sequencer that handles all four different data formats. Although each control unit is now more challenge to design, the unified datapath reduces the number of control units.

This sharing of resources between the CPU and FPU boosts the usage of on-chip logic while maintaining full functionality, reduces die area and power dissipation. This innovative combination contributes significantly to the R4200 in achieving 55 SPECint92 and 30 SPECfp, under 2 watts of power consumption with die size of 8.8mm x 8.6 mm. Refer to Table 1: Integer and Floating-point instruction latencies for detailed performance of each class of integer and floating--point instructions.

The only compromise for a unified datapath design such as this is that long-latency floating-point instructions cannot proceed in parallel with the execution of other ALU instructions. This leads to a moderate decrease in floating-point performance as measured by the SPEC benchmark, but this fact does not unduly affect the performance of the R4200's target applications which require integer performance. With this unified datapath design, all integer instructions except multiply and divide remain single-cycle operations.

Acknowledgments

The authors would like to thank the entire R4200 team for their efforts in bringing this project to fruition, as well as their invaluable help with this paper. We would also like to recognize the efforts of the CAD and Engineering Computer Services personnel, without whose efforts this project could not have been completed.

Also, the authors would like to thank the semiconductor partner NEC for their contributions in making this product a success.

References

[1]: Hennessy, J., N. Jouppi, F. Baskett, and J. Gill, "MIPS: A VLSI Processor Architecture," Tech. Report 223, Computer Systems Laboratory, Stanford University, Stanford, Calif., 1983.

[2]: ANSI/IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Architecture, IEEE, 1985.

[3]: R4200 Microprocessor Preliminary Product Information, MIPS Technologies Inc., 1993.