CASE STUDY OF THE PIPELINED ARITHMETIC UNIT FOR THE TI ADVANCED SCIENTIFIC COMPUTER

Charles Stephenson
Texas Instruments Incorporated
Austin, Texas

Introduction

Many scientific applications today require computers which are very fast and capable of processing large amounts of data. Some advances in scientific processing have been slowed due to the lack of supercomputer capabilities which are required primarily in the area of Central Processor speed and the availability of large amounts of high speed memory. Particularly in the fields of modeling and simulation, additional speed and memory capacity are desired to allow increased resolution of the experiment. Technological developments in such things as integrated circuits, multilayer printed circuit boards, memory speeds, and others have contributed to the ability of computer manufacturers to serve this market. In addition to these developments, however, large advances had to be realized from the standpoint of the basic computer architecture. The concept of pipelining has provided an answer to the large data execution rate required. Pipelined capabilities in the form of arithmetic units and special purpose functional units are included in machines such as the CFC7600, IBM 360/195, CDC STAR-100, etc.[1,2] The Texas Instruments Advanced Scientific Computer (ASC) uses the pipeline concept throughout the Central Processor and carries the concept throughout the Central Processor and carries the concept further to include vector instructions in response to the high execution rates required.[3]

The ASC Central Processor is composed of three kinds of units as is indicated in Figure 1.[4] The Instruction Processing Unit (IPU) fetches instructions, decodes the operation code, develops the address of the memory operand, and resolves address hazards. Forty-eight addressable registers are resident in the IPU. The Memory Buffer Unit (MBU) has complete control during vector instructions and calculates all memory addresses required for vectors. (A discussion of vector instructions is contained in the following section titled "Definitions"). The MBU contains a Read-Only Memory (ROM), which provides the basic control for the Arithmetic Unit (AU). The AU receives all operands from the MBU and provides results to either the IPU or MBU. The ASC can be configured with one or two IPUs, each supplying operations for one, two, three, or four identical MBU/AU pipelines. Seven ASC systems have been manufactured; five with one pipeline each, one with two pipelines, and one with four pipelines.

Definitions

Pipelining has been defined as a technique of imbedding concurrency in a computer system by implementing it in the form of a pipeline, a configuration of independent autonomous units each of which is dedicated to perform a specific subfunction in an overlapped mode with others.[5] In terms of ASC nomenclature, the pipeline is a series of sections, each of which performs an independent operation. The requirement that each section must be independent requires that any information calculated in one section which will be used in succeeding sections must be captured in a register or latch, and that output is transmitted to the next section. A pipeline can have a fixed configuration or the configuration can be variable which is under hardware control. The ASC is implemented with a fixed configuration in the Instruction Processing Unit and a variable configuration Arithmetic Unit[6]. Figure 2 represents the IPU in pipeline form. Each instruction passes through the sections in the order indicated and the appropriate operation is performed. The remainder of this paper will deal primarily with a description of the variably configured pipelined Arithmetic Unit.

Other terms which require definition are Vector and Scalar instructions. A scalar instruction is one which operates on only one or two operands (typically one register operand and one memory operand) to produce one result which normally would be placed in the addressed register. A vector instruction is one which operates on one or two series of operands (usually two different areas of memory) and produces a series of results (or, in special cases, a single result). Examples of this would be a Vector Add which would take n elements of Vector A, add those to the corresponding n elements of Vector B, and produce a result Vector C which has n elements. A Vector Dot Product (VDP) instruction, however, can multiply A_n by B_n and produce a single result which represents the sum of the individual products. When executing vector instructions, the basic measurement, or the effective vector rate, is stated in clocks per result. The goal with a single pipeline is to operate upon a new set of operands each clock. For vectors which produce n elements, an effective vector rate of one clock per result would require n clocks plus some time to fill the pipe. If an instruction uses the same pipe section for two clocks without allowing the next operand to enter, the vector rate is two clocks per result.

The clock rate required by a pipeline is controlled by the slowest section, or bottleneck. Obviously the design goal is to implement the individual pipe sections such that the propagation delays are approximately equal. The amount of logic performed in the individual sections will then determine the length of the pipeline. Figure 3 represents three possible pipeline structures for performing a Floating Point Addition. In each case, the following functions must be performed: (1) Determine the difference in the two exponents, (2) shift the smaller operand to the right to align the fractions, (3) add the fractions, and (4) normalize the result. Each of the configurations can yield a vector rate of one clock per result so the optimum configuration when considering only vector execution rates would be one which contained many simple pipeline sections with the resultant increase in clock rate since the logic has been overlapped. However, when considering only scalar execution rates the optimum configuration would be one containing very few sections.

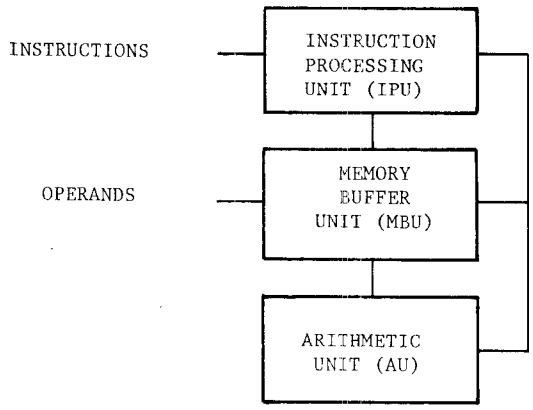

FIGURE 1

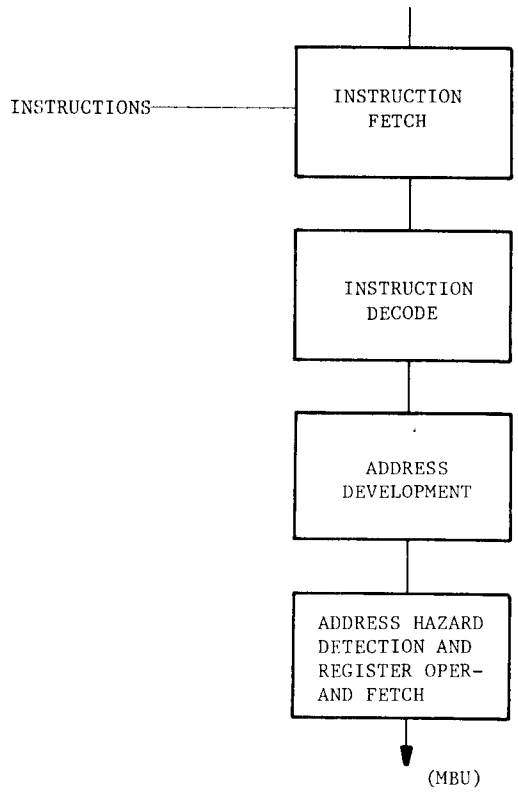

FIGURE 2

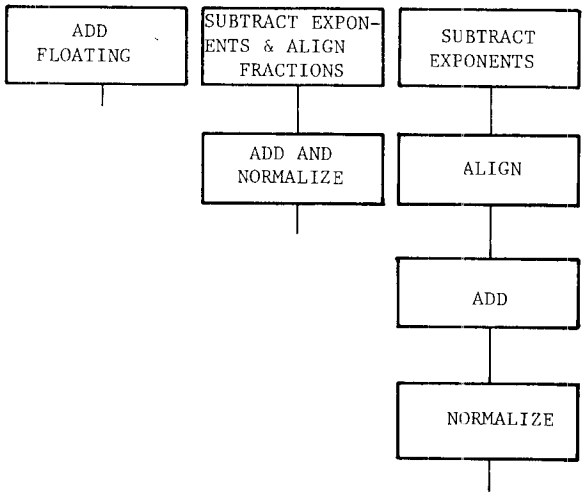

FIGURE 3

The ASC Arithmetic Unit

One of the key design considerations of the AU was the approach used to control the execution of a large instruction set. As was stated earlier, the basic control is with a ROM located in the MBU. The resultant MBU/AU pair forms a pipeline which contains all the necessary hardware to perform instructions supplied by the IPU. In particular, all vector instructions fetch operands and store results with no further intervention from the IPU. Therefore when reference is made to the AU, actually the "pipe" is a more representative term. Some of the basic requirements in designing the Arithmetic Unit for the ASC were:

1. Vectors performed at one clock per element.
2. Efficient scalar execution.
3. Very fast Vector Dot Product.
4. Fixed- and Floating-point formats.
5. Synchronous clock at best speed.
6. 64-, 32- and 16-bit word sizes.
7. Flexible control for each of modification.

An overriding factor which is, of course, present in all designs which ultimately are implemented is to keep costs down. Thus, it was decided to execute scalar and vector instructions with the same hardware. This along with requirements 1 and 2 listed above resulted in the implementation of a pipeline to achieve one clock per element but a pipeline which consisted of as few pipe sections as practical to not unduly penalize scalar execution. A good example of this type of trade-off can be seen by examining a shift instruction only. The shift hardware must be able to shift a 64-bit word n places. A single pipe section as shown in Figure 4 can be implemented which contains all of the decode logic and selection trees for all bit positions. This represents several levels of logic plus the set-up time required to appropriately load the holding register at the output. Conversely a pipeline which consists of six sections as shown in Figure 4b will perform a vector shift at a rate of one clock per result where the clock rate can be approximately the set-up time for the output register since no logic is performed and only hardwired propagation times are involved. Obviously the vector rate is best with many pipe sections but the scalar rate will degrade. Even more significant amounts of hardware are involved and sharp differences in execution times are realized when making the same type of tradeoff for instructions such as multiplication and division.

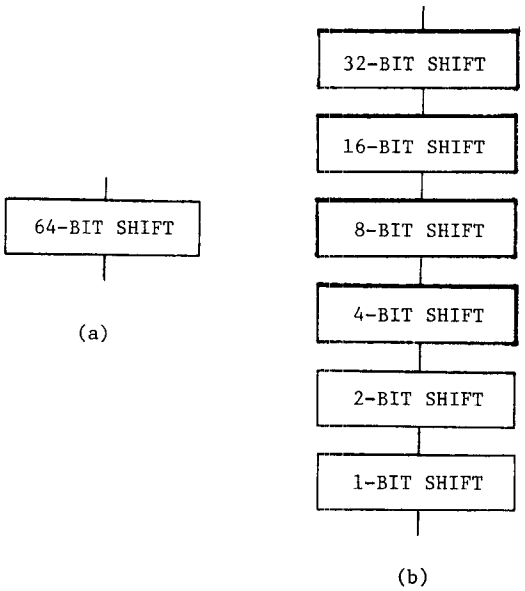

FIGURE 4

Figure 5 represents a block diagram of the ASC's AU. There are eight distinct sections. These sections were all implemented within ten levels of Emitter Coupled Logic (ECL) integrated circuits which was the design goal. The particular choice of pipeline sections were incorporated to satisfy requirement (3) from above - to implement a very fast Floating Point Vector Dot Product (VDP). By examining this instruction, the general use of the AU can be understood. For this vector, the MBU must provide two operands on each clock. As with all vectors, the MBU performs look-ahead operand fetches on both an A vector ($\vec{A}$) and a B vector ($\vec{B}$). The MBU requests 8-word memory blocks, referred to as octets, from an 8-way interleaved memory system which is normally implemented with 160 nanosecond BiPolar Memory. Since the MBU has total control of the vector instruction it is able to perform 3-octet look-ahead on each operand stream. The Input Section of the AU serves as a means of buffering data from the MBU and also provides a means of routing previous results directly back into the AU for execution without being resident in the IPU's Register File (this is referred to in the ASC as a "short-circuit"). The Floating Point VDP passes through the AU in the following way: (1) Multiplier, (2) Accumulator, (3) Exponent Subtract, (4) Align, (5) Adder, (6) Normalizer. Partial products circulate through these sections until a final result is obtained which then passes through the Output section before being transmitted to the MBU. A brief description of each section follows which relates the function of each section in the execution of the VDP and how other instructions use these same sections.

The Multiplier section is capable of multiplying two 32-bit numbers and producing a 64-bit result. In the case of the Floating VDP the operands are sign and magnitude with the seven exponent bits being automatically ignored. However, this section is also used for fixed point multiplications with the fixed point format being two's complement. In addition, all divide instructions are performed as interactive multiplications and thus use this section of the pipeline. The hardware consists of a Wallace-type summand tree of full adder circuits with recoding of the multiplier operand. The output of this section consists of two 64-bit registers, the Pseudosum and Pseudocarry which must be added together to obtain any multiplication result. Thus, no final result ever passes from the multiplier to the Output Section but must always pass through the Accumulator.

The Accumulator section is a 64-bit carry-propagating adder implemented using a double-level lookahead for carry generation. For the Floating VDP, this section merely produces the partial products by adding the outputs of the Multiplier section. The partial product obtained is one of the two operands that will be routed to the Exponent Subtract section. The Accumulator is able to add three 64-bit numbers together with the third operand being the output of the Accumulator fed back for performing the accumulation of partial products for the fixed-point VDP and also for all Double-Length multiplications or divisions which are performed by making iterative passes through the Multiplier and Accumulator.

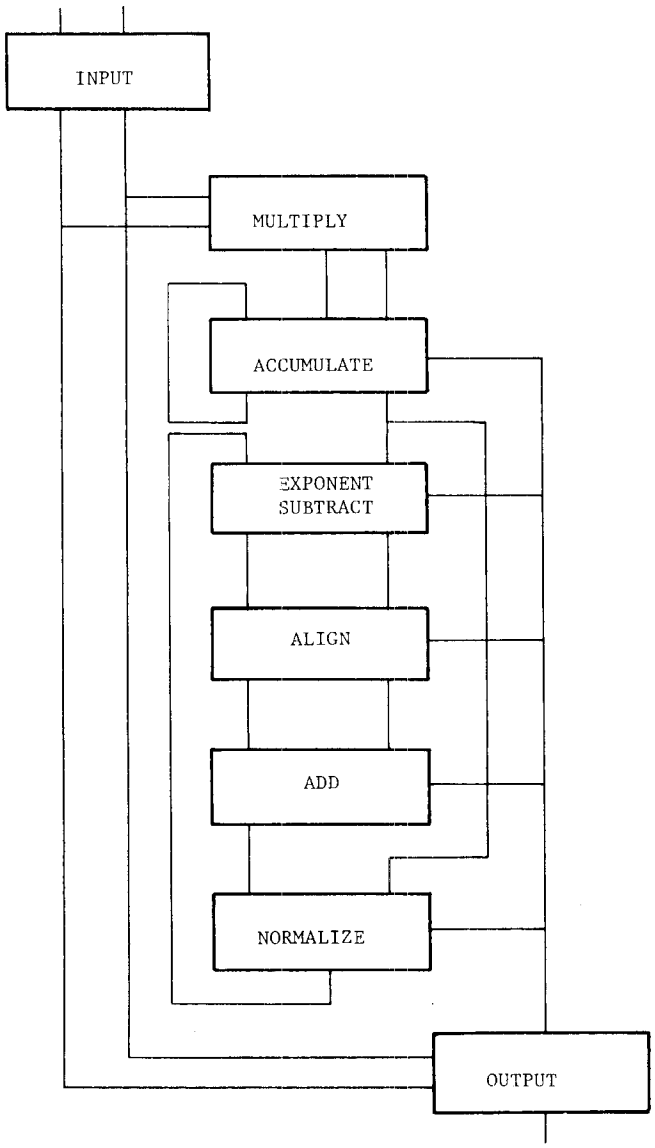

FIGURE 5

The exponent Subtract section operates upon the floating point partial product which is at the output of the Accumulator section and a partial product which is at the output of the Normalizer section to determine the smallest operand and how many bit positions that operand must be shifted to align mantissas. Since the floating point numbers are represented by a hexadecimal exponent, the shift count will indicate a four bit increment from zero to fifty-six. For all other instructions which use the exponent subtract section, the operands are supplied by the Input section. The Exponent Subtract section contains the comparison logic for all fixed- and floating-point compare instructions, including all Vector Compares. The major output of this section is the Large Operand Register and the Small Operand Register to be supplied to the Align section.

In the Align section, all floating-point and add-type instructions (Floating Add, Subtract, Add Magnitude, etc.) align the mantissas by shifting the Small Operand Register to the right in one clock the appropriate number of hexadecimal positions. For the Floating VDP this function is being performed at the same time that the next pair of operands are being compared in the Exponent Subtract section. The Align section performs all shift instructions that require a right shift. Shift instructions require a bit shift of zero to sixty-four and include logical, arithmetic, and circular shifts. To reduce the amount of logic required in this section, the implementation uses two clocks to complete. On the first clock, a hexadecimal shift is completed from zero to sixty bit positions. On the second clock, a bit shift of zero to three occurs which completes the instruction. This implementation results in fixed-point shift instructions having a two clock per result execution rate.

The next operation performed for a Floating VDP is the addition of the aligned operands. The Add section contains a sixty-four bit carry propagation adder with double-level lookahead for carry generation. Inputs are selected from the Align section for all floating-point additions, subtractions, etc. The result is placed in a single register which represents the result of the addition. As stated above, for the VDP, the Add section operates independently upon one partial product while the other sections are operating on other partial products.

Since all floating-point results are required to be hexadecimally normalized, the output of the Add section is selected by the Normalize section where leading zeros are examined to determine how far to shift the result to the left to achieve normalization. In the same manner that the Align section performed right-shifts, the Normalizer executes all left shifts. Once a partial product of the Floating VDP is normalized, it is ready to re-enter the Exponent Subtract section to be combined with the partial product leaving the Accumulator. It can be seen that four pipe sections exist between the point where an individual partial product (An X Bn) leaves the Accumulator and the time that a floating-point addition is completed at the output of the Normalizer $[(A_n X B_n)+(A_{n+4} X B_{n+4})]$. This results in a continual circulation of four partial products in the pipeline, the MBU controls the sequences of the partial products through the sections in such a manner as to produce the final result.

The Output section receives results from the appropriate section depending upon the instruction performed and sends the output to either the MBU or IPU. The Output section contains hardware to perform logical instructions such as And, Or, Exclusive-Or, etc. The operands are received directly from the Input section for these instructions.

Vector and Scalar Control

In addition to implementing a small number of pipe sections to optimize scalar execution, while providing vector streaming, the control is designed to reconfigure the pipeline for each instruction, thus minimizing the number of sections utilized for a given instruction. Figure 6 demonstrates different configurations which are used for (a) a fixed-point addition, (b) a floating-point addition, and (c) a fixed-point multiplication. The ROM which controls the AU has 512 addresses by 256 output lines. Some of the ROM output is used by the IPU and MBU with approximately 180

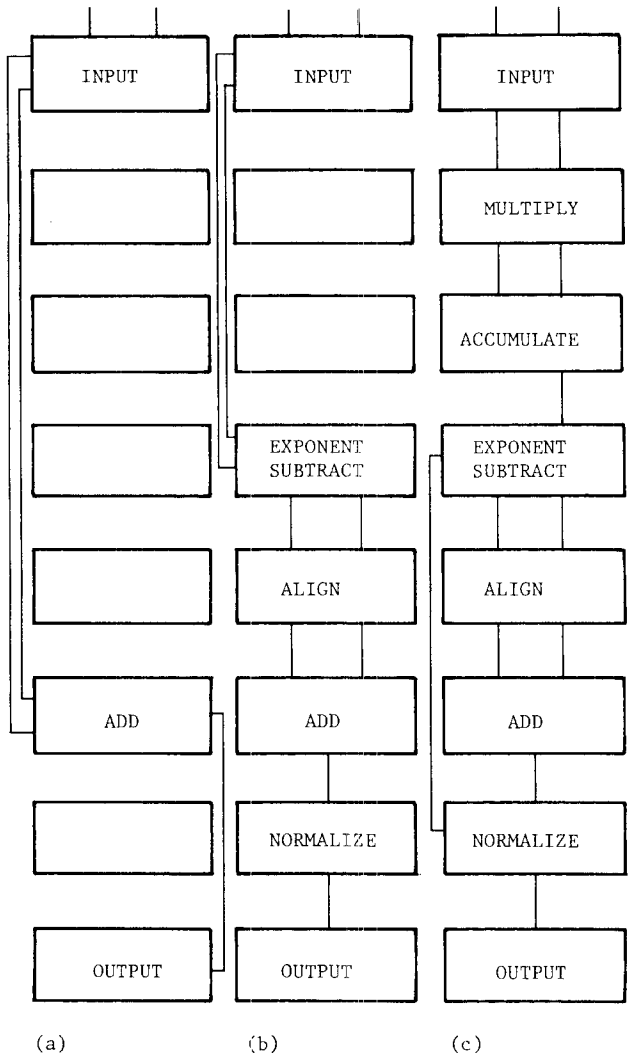

FIGURE 6

individual signals being routed to the AU. Each scalar instruction produces signals that control the functions to be performed in the AU and performs the timing by stepping through a fixed sequence. Only one instruction-type is resident at any one time in the internal sections of the AU and the ROM output is enabling signals necessary to that instruction. All sections of the pipeline are operating at each clock as could be determined by examining the "don't care" conditions but their output register is merely not selected by an adjacent section. The number of internal sections required by an instruction constitutes the number of ROM locations required. With respect to Figure 6, the number of ROM locations are one, four, and two respectively.

For vector instructions, once the pipeline is filled, the configuration remains fixed until the MBU recognizes that the addresses corresponding to the end of each loop has been fetched. At that time, the pipe must be reconfigured to complete the instruction such as adding together the four partial products at the end of the floating VDP. This is accomplished by having two addresses (B_1 and B_2) provided as output signals from the ROM. These addresses feed back into the input of the ROM as controlled by the MBU, B_1, being normally selected with B_2 chosen when end-of-loop is observed. For example, a vector may sequence through location 100, 101 and 102 until the pipe is filled. Once the pipe is filled the control bits do not change for the entire loop length thus address location 103 in addition to providing the control bits yields a ROM address B_1 equal to 103 and B_2 equal to 104. When end-of-loop is reached B_2 causes the ROM to go to 104 and then sequence through remaining addresses until the pipeline is empty.

A primary reason for selecting ROM as the control mechanism was the ease of correction, addition, and maintainability of instruction. A special ROM card tester is used to program the ROM packages to the patterns described in the logical implementation. When changes are desired the card tester indicates the packages to be replaced and creates the new patterns in the replacement packages. Using this technique, many implementation errors were corrected during integration. Also, new algorithms have been incorporated to speed up instructions such as Divides and VDPs with short size loops. In addition, new instructions (Select, Replace, and Map) have been implemented.

Summary

In summary, the Texas Instruments ASC was designed to use Arithmetic Unit pipelines that achieve a vector rate of one clock per result. To accomplish a goal of efficient scalar executions, a method of reconfiguring the pipeline for each instruction was required. The instruction set, and in particular the design goal of a very fast floating-point Vector Dot Product produced a minimum requirement of eight pipe sections. Design constraints in the Instruction Processing Unit required the use of ten logic levels which became the upper limit throughout the pipeline. It was determined that both fixed- and floating-point operations could be performed within that number of levels; and thus, cost could be reduced by sharing the logic. The selection of a ROM for the AU control has allowed new vector instructions to be added to the repertoire after the hardware was built and has provided a flexible control environment which has proven to be easily documented and maintained. The Arithmetic Units are installed with machines in Austin, Texas; Amstelveen, Holland; Huntsville, Alabama; and Princeton, New Jersey in both single and multiple pipeline systems. The units are executing the full ASC instruction set and have proven to be very reliable.

References

1. Anderson, D.W., F.J. Sparacio, and R.M. Tomasulo, "The IBM System/360 Model 91: Machine Philosophy and Instruction Handling", IBM Journal of Research and Development, Vol. 11, No. 1, January 1967, pp 8-20.

2. Purcell, Charles J., "The Control Data Star-100- Performance Measurements", Compcon 1974.

3. Watson, W. J., "The TI-ASC - A Highly Modular and Flexible Super Computer Architecture", AFIPS, FJCC, 1972, pp. 221-230.

4. Watson, W.J. and H.M. Carr, "Operational Experiences with the TI Advanced Scientific Computer", Compcon 1974.

5. Remamoothy, C.V. and H.F. Li, "Efficiency in Generalized Pipeline Networks", Compcon 1974.

6. Stephenson, C.M., "Control of a Variable Configuration Pipelined Arithmetic Unit", Allerton Conference on Circuit and System Theory, October 1973.