Special issue of the 2nd International Conference on Computational and Experimental Science and Engineering (ICCESEN 2015)

Accelerating Processors with Simple Touches

A. Gürhanli*
Bursa Orhangazi University, Department of Computer Engineering, Bursa, Turkey

Computer architects are familiar with complex processor designs. A tiny performance improvement may need involvement of numerous experts and months of tedious work. But sometimes without adding lots of extra hardware or inventing complex hardware algorithms, it is still possible to accelerate a design substantially by doing minor alterations in the data path. This paper presents how an ARM compatible processor's clock rate became 1.285 times faster by changing IO pads, employing multiplexers, and improving the ALU design. The processor has been implemented in Verilog HDL and the performance improvements have been verified by simulation using the Design Compiler tool of Synopsys.

DOI: 10.12693/APhysPolA.130.252
PACS/topics: 07.05.Bx, 89.20.Ff

1. Introduction

Amdahl's law states that the overall speedup obtained by improving some portion of a computer system is limited by the portion of the time during which the faster mode can be used. The law is described with the formula below,

$$S_{\text{overall}} = \frac{1}{(1 - F_{\text{enhanced}}) + \frac{F_{\text{enhanced}}}{S_{\text{enhanced}}}}, \tag{1}$$

where S is the speedup and F is the fraction of time during which the enhanced speedup can be used.

Hennessy [1] gives an interesting example. Overall speedup obtained from a marvelous design that makes the floating point square root unit 10 times faster will be only 1.22 times, if the square root operation is responsible of 20% of the overall execution time. The overall speedup will be even less if enhanced fraction is smaller.

In recent years, the main focus of computer architecture researches is on multicore processing and parallel computing. However, a vast majority of embedded processors still have only one core and their speed cannot be improved with multicore approaches. So decreasing the average clock cycles per instruction and increasing the clock frequency are still major acceleration methods for single core embedded processors. Multicore processors will benefit from clock frequency improvements, as well.

When the clock frequency of a processor is increased, all the instructions are related with the improvement and the F_{enhanced} term in Eq. (1) is 100%. That will yield $S_{\text{overall}} = S_{\text{enhanced}}$. So optimizations and design improvements related to clock frequency are worth to care about.

2. Related work

Numerous academic researches have been done with the goal of having faster processors. Some focused on improving clock frequency, some others proposed architectures crunching more instructions per clock cycle, and recent researches mostly dealt with exploiting potential parallelism. Various instruction-level, thread-level and request-level parallelism opportunities have been evaluated by many research groups.

For example, Albonesi [2] introduced a dynamic IPC/clock rate optimization. Childers [3] developed an adaptive processor supply voltage for instruction-level parallel schemes. Hartstein [4] investigated the optimum pipeline depth for a microprocessor. Hsu [5] proposed a compiler directed dynamic frequency and voltage scaling mechanism. Iyer [6] evaluated power and performance of globally asynchronous locally synchronous processors. Milutinovic [7] worked on pipeline design tradeoffs on the basis of a 32-bit gallium arsenide processor. Adl-Tabatabai [8] focused on unlocking concurrency for multicore programming with transactional memory. Powell [9] tried to reduce set-associative cache energy by employing selective-direct mapping and way prediction.

Esmaeilzadeh [10] showed that the speedup from using multithreading on one core on an i7 processor averaged at 1.28 for the Java benchmarks and at 1.31 for the PARSEC benchmarks. This result tells us how hard it may be to obtain a tiny performance improvement even by employing complex and expensive methods like multithreading.

Our economic optimizations for an ARM instruction-set-compatible embedded processor yielded a performance improvement of 1.285 times, which is comparable to multithreading benefits. This paper uses the master thesis outcomes of the author [11].

3. Original architecture

This work has been based on an ARM instruction-set-compatible RISC processor. The processor has been implemented using the Verilog HDL. The original ARM 7 architecture is given in Fig. 1.

The processor is pipelined into three stages; fetch, decode and execute. Fetch stage captures data from

*e-mail: ahmet.gurhanli@bou.edu.tr

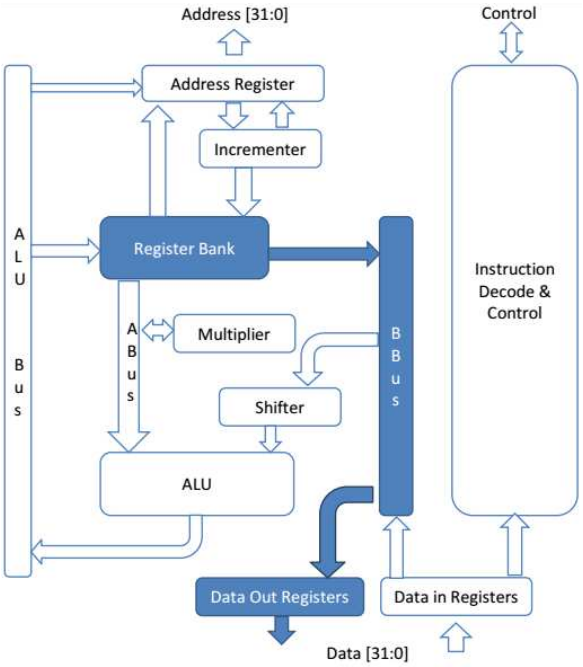

Fig. 1. Original architecture of an ARM 7 processor. The critical data path, limiting the clock frequency, is the path for writing data to the memory.

the memory. Decode stage interprets the instruction and generates the control signals for execution stage. In the execution stage the data is processed according to the signals coming from the control unit. Decode and control unit is separated from the execution stage by pipeline registers. The main components in the execution stage are register bank, shifter, ALU, incrementer and multiplier.

Register bank has two input and three output ports. One input and one output port is dedicated to program counter, which is one of the 16 user registers. Shifter and ALU are connected serially. So the operands can be shifted before being fed into ALU. Shifter can perform logic and arithmetic shifts both towards left and right.

ALU performs 16 different arithmetic operations and several logical operations. Address register may be loaded from three sources; ALU bus, program counter or incrementer. There are two registers for instruction reading and one for data reading. The data read from memory or to be written to memory can be a word or a byte. Both big-endian and little-endian formats are supported.

4. Design enhancements

After the first synthesis with Synopsys Design Compiler, the critical path for timing was the store path as shown in Fig. 1. This path was made faster by employing a different output pad. The technology used (UMC 0.18 CMOS) has two different pads, one has a better speed and the other one has lower noise. When the faster pad (P2C) was chosen, store path was no longer the bottleneck for a faster clock.

After changing the I/O pads, the critical path became the one from data-in registers to the register bank, as shown in Fig. 2. This path covers data in registers, B bus, shifter, ALU, ALU bus and register bank.

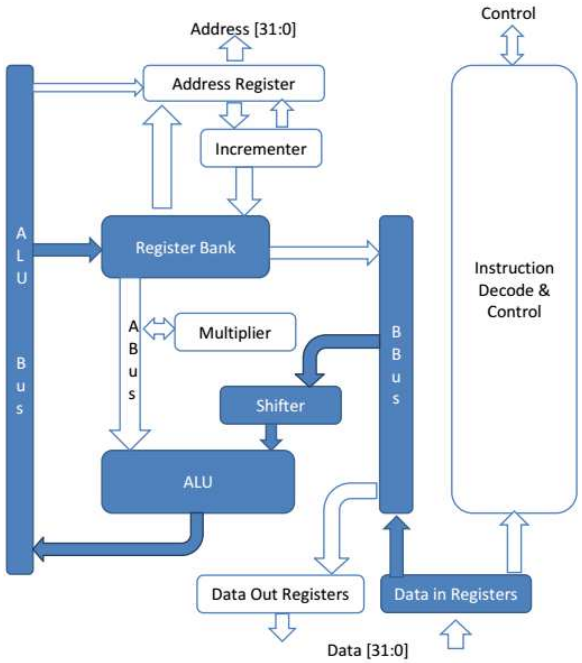

Fig. 2. Second critical path. Moving data from memory to the register bank involves numerous data path units, including data-in registers, B bus, shifter, ALU, ALU bus and register bank.

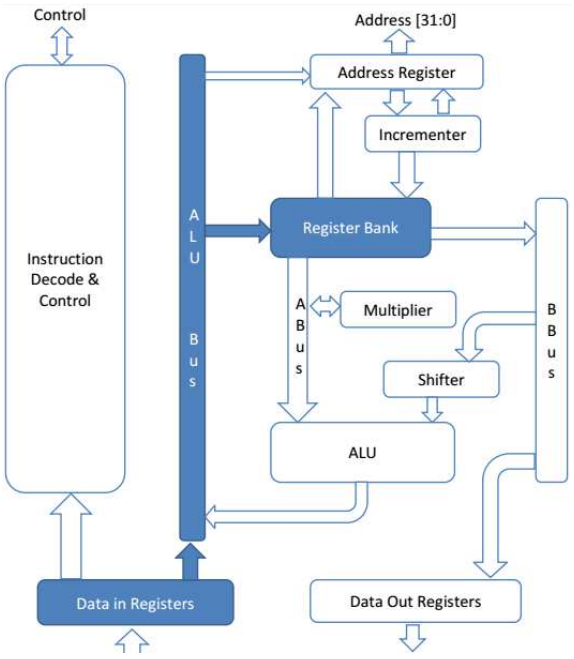

Fig. 3. Modified shorter read path. A load instruction does not need shifter and ALU units, because memory operations and data processing are separated.

It can be seen that the read data does not need to pass through shifter and ALU. So, in order to minimize the length of this path the read data is given to ALU bus directly as shown in Fig. 3. After doing this modification, this path was no longer the critical path.

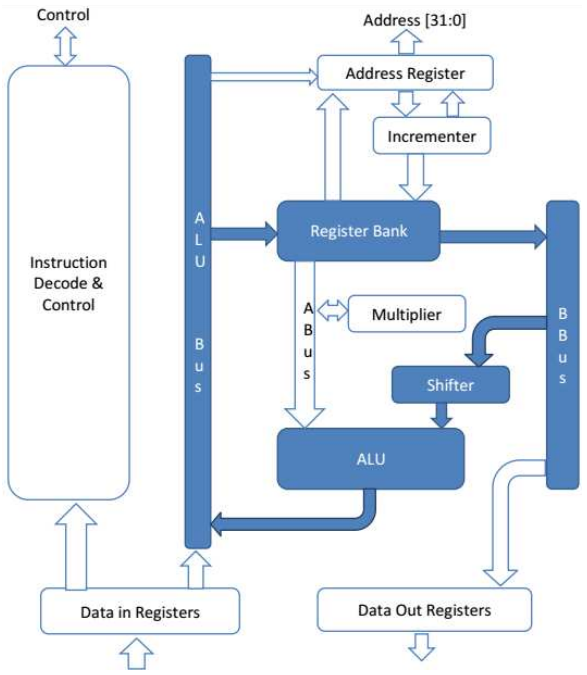

Fig. 4. The final critical path is the one starting from register bank and passing through B bus, shifter, ALU, and ending up in the register bank again.

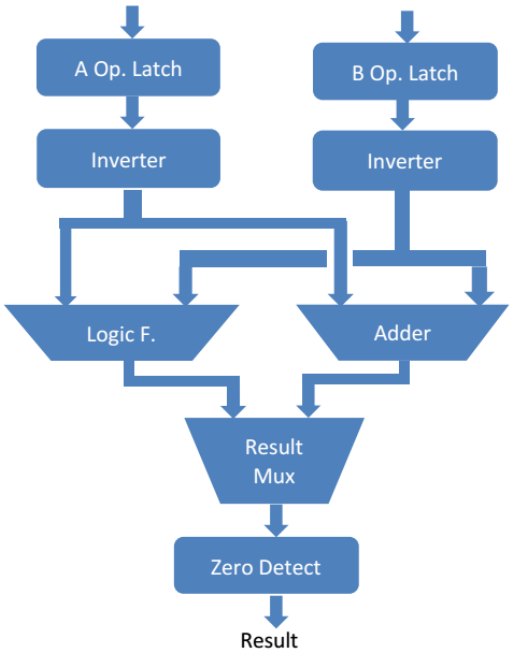

Fig. 5. Original ALU structure. Latches at the entry hold the operands. An inverter functions serially after the latches.

The final critical path is the one starting from register bank and passing through B bus, shifter, ALU, and ending up in the register bank again, which is shown in Fig. 4. In order to shorten this path, the architecture of the ALU was modified, as shown in Fig. 5. The serial latches and inverters were removed and all components were made parallel, with the exception of the multiplexers and routers as shown in Fig. 6.

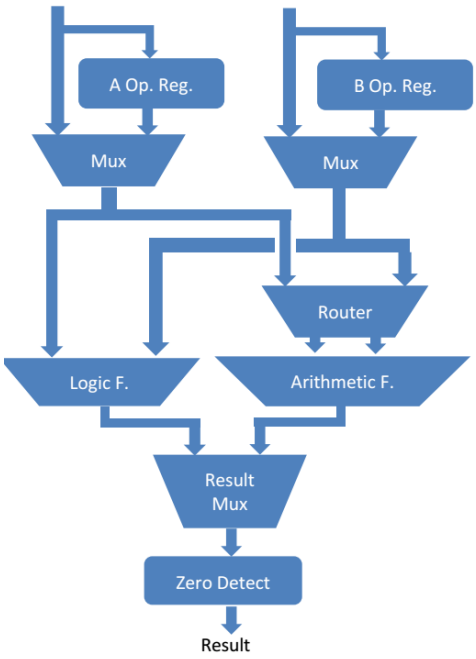

Fig. 6. Modified ALU. Serial latches and the inverter at the entry have been removed. Registers keep operands for next clock cycle only when needed. Inverting operation is done at the arithmetic functions unit.

5. Conclusions

After making changes stated above and giving tighter timing constraints to both synthesis and place-route tools, the highest frequency is increased from 70 MHz to 90 MHz. This speedup of 1.285 times is at the level of speedups obtained from complex and expensive techniques such as multithreading. When developing complex hardware for exploiting potential parallelism in the program execution, it might be very useful to evaluate the possible design improvements in the data path that will lead to a faster clock frequency.

References

[1] J.L. Hennessy, D.A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed., Morgan Kauffman, Waltham 2012.

[2] D.H. Albonesi, in: *25th Intl. Symp. On Computer Architecture*, IEEE, Barcelona 1988, p. 282.

[3] B.R. Childers, H. Tang, R. Melhem, in: *Kool Chips Workshop*, 2000, p. 78.

[4] A. Hartstein, T.R. Puzak, in: *29th Intl. Symp. On Computer Architecture*, IEEE, Anchorage 2002, p. 7.

[5] C.-H. Hsu, U. Kremer, M. Hsiao, in: *Workshop On Power-Aware Computer Systems*, IEEE, Huntington Beach 2000, p. 275.

[6] A. Iyer, D. Marculescu, in: *29th Intl. Symp. On Computer Architecture*, IEEE, Anchorage 2002, p. 158.

[7] V. Milutinovic, D. Fura, W. Helbig, *IEEE Trans. Comp.* **40**, 1214 (1991).

[8] Ali-Reza Adl-Tabatabai, C. Kozyrakis, B. Saha, *Unlocking Concurrency: Multicore Programming With Transactional Memory*, Acm Queue, 2006.

[9] M. Powell, A. Agrawal, T.N. Vijaykumar, B. Falsafi, K. Roy. in: *34th Intl. Symp. On Microarchitecture*, IEEE, 2001, p. 54.

[10] H. Esmaeilzadeh, T. Cao, X. Yang, S.M. Blackburn, K.S. Mckinley, in: *The ACM International Conference On Architectural Support For Programming Languages And Operating Systems*, ACM, New York 2011, p. 319.

[11] G. Ahmet, Chen Charlie Chung-Pin, M.Sc. Thesis, Graduate Institute Of Electronics Engineering, National Taiwan University, 2006.