Improving a fixed-point RISC processor by a hybrid adder

S.A.M. BARANDAGH†, H.S.SHAHMOSEINI‡, N.BABAII RIZVANDI†
†Iran Telecommunication Research Center(ITRC),
EDA laboratory, End of North Kargar St., P. O. Box 14155-3961,
Tehran 14399,
IRAN.
Tel/Fax : +98 21 8497438,
‡Iran University of science and Technology, Narmak,
Tehran, 16884, IRAN.

Abstract :-The large propagation delay in a digital circuit reduces it’s performance. Detecting the longest delay or critical path within the circuit and improving the hardware which causes the delay is an standard method for increasing the performance of design. In this paper, a previously designed RISC processor is improved by finding one of the longer paths which was located in ALU, a substituting the adder of ALU by a high–speed hybrid adder. After improvement, simulation and synthesizing of the processor are performed. The simulation results show that the processor can work at 37.81 MHz, i.e. 45.6% better than before.

Key-Words: - RISC Processor, Structural Model, MIPS.

1. Introduction
The MIPS is known as one of the best RISC processors ever designed, and grew out of research started at Stanford University. It came in about 1987, from the MIPS project. Along with the Berkeley RISC projects, the Stanford MIPS project was one of the first publicly known implementations of the Reduced Instruction Set Computer (RISC) architecture. Among all the current RISC architectures, MIPS remains as one of the simplest. This simplicity makes the MIPS architectures a favorite choice among universities and research labs for using as the base of their design. This simplicity also makes the MIPS architecture very attractive to the embedded microprocessor market as it enables very cost-effective implementations.

A MIPS-based RISC processor was introduced in [1]. It was a fixed-point processor and consist five stages, i.e., fetch, decode, execute, memory access, and write back. The most important feature for evaluating a design is maximum attainable frequency. It can be found by detecting the longest delay or critical path within the different stages of the processor. In previously designed processor the critical path was located in ALU stage. In this paper, the previous ALU is substituted by another one, which employs a hybrid adder.

The rest of the paper is organized as follows: in section 2, the previously designed architecture is described briefly. In section 3, ALU, which affects the frequency of the processor, is explained. In section 4, a high-speed adder is described. The performance of the processor is evaluated in section 5. In section 6 the paper is concluded.

2. Previous design
MIPS design was intended to simplify processor design by eliminating hardware interlocks between the pipeline stages. A MIPS processor consists of an integer processing unit and a collection of coprocessors that perform ancillary tasks or operate on other types of data [3]. Like other RISC designs, the MIPS instruction set is straightforward. MIPS is a load/store architecture, which means that only load and store instructions access memory. Other instructions can only operate on values in registers [2-5].

Generally, the MIPS instructions can be broken into three classes: the memory-reference instructions, the arithmetic-logical instructions, and the branch instructions. Also, there are three different instructions formats in MIPS architecture: R-Type instructions, I-Type instructions, and J-Type instructions [6].

In this section, the processor which was introduced in [1], is described.

For describing the processor architecture, a basic model is chosen. Figure 1 represents the top-level schematic of the processor. This schematic shows the principal components, or main blocks of the processor.

The processor has following features:
(i) The MIPS architecture is included 47 instructions.
(ii) There are five stages in the pipeline (i.e. instruction fetch stage, instruction decode/register fetch stage, instruction execution stage, memory access stage, and write back stage).
(iii) The register file has thirty two 32-bit registers.
(iv) This processor has a 32-bit address bus and a 32-bit data bus.
The pipeline structure of the processor is a modified version of a popular load/store RISC [3]. The five stages of the pipeline can operate concurrently, using synchronization signals: Clock and Reset. When an instruction is fetched at the first stage, it is read from memory using the address in the PC and then placed in the IF/ID pipeline latches. There are two subunits in the fetch stage: PC and PC-Increment. Table I shows the subunits of the different stages.

Table I: The subunits used in the different stages of the processor.

<table>
<thead>
<tr>
<th>Stage number</th>
<th>Subunits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage 1</td>
<td>PC, PC_Increment, PC_Mux, Hazard_detection_Unit</td>
</tr>
<tr>
<td>Stage 2</td>
<td>Main_Control_unit, Jump_control, Ctrl_Mux, Register_file_32_32, IR_Discrim, Extender, Shamt_mux</td>
</tr>
<tr>
<td>Stage 3</td>
<td>Course_register, EPC, 32_adder, Except_Mux, ALU, ALU_Control, J_I_control, ALU_Mux, Forwarding_Unit, Mux_Reg</td>
</tr>
<tr>
<td>Stage 4</td>
<td>Branch_Mux, Branch_Control, Bus_fsm_Ctrl, Store_extension, Load_Extension</td>
</tr>
<tr>
<td>Stage 5</td>
<td>Mux_MemtoReg,</td>
</tr>
</tbody>
</table>

In the memory access stage, load instruction reads the data using the address from the EX/MEM pipeline latches and loads the data into the MEM/WB pipeline latches. The memory access stage executes load and store instructions. In this stage, the program can access 32-bit words of data in memory and load them into a general purpose register, or it can be stored the words to memory from a general purpose register. This stage contains a multiplexer and three functional units: store and load extension unit, branch control unit and branch mux. At the final stage, write-back stage, the data is read from the MEM/WB pipeline latches and written into the register file. This stage has two multiplexers. In addition to the five mentioned pipeline stages, there are other sub-blocks in the processor.

The data and control hazard problem in the processor are resolved by forwarding and making an special hardware respectively.

3. ALU

During execution cycle, the processor performs logical, arithmetical, shift and comparison operations on the data in fields of the instructions and put them on the appropriate latches. At the execute stage, for I-type and R-type instructions, the contents of a register is read. Then immediate part of the instruction which had been sign-extended in previous stage, is added with the register contents by ALU. The result of the addition is placed in the EX/MEM pipeline latches. There are three functional units in the execution stage of this model: ALU, ALU-control, and forwarding unit. There are also other sub-blocks such as exception program counter, and multiplexers. During the execution cycle, ALU performs operations. Load and store instructions calculate absolute addresses of data, and jump and link instructions calculate return addresses in this cycle.

Figure 1: Top-level schematic of the processor containing 47 instructions.

At the decode stage, the 16-bit of the immediate field of the instruction is sign-extended to 32 bits, and two registers are read from register file. All three values are stored in the ID/EX pipeline latches, along with the incremented PC address. The decode stage contains five subblocks: main control unit, register file, jump control unit, two extender and shift amount units. During the instruction decode, the IR_Discrim separates different
ALU input registers. These operations are controlled using two lines from ALU-control block. By means of this lines, four parallel blocks: logical, arithmetic, shifter, and comparison are identified (Figure 3).

Then the internal sub-blocks of the above four blocks are controlled using these lines. Logical and arithmetic blocks performs logical and arithmetic operations such as AND, OR, XOR, etc and as ADD, SUB, ADDI, etc respectively.

The adding function is fundamental to determining processor cycle time and hence overall performance. Addition algorithms have been widely studied, so that there are many apparently different algorithms that actually differ only in some detail. For designing a high-speed adder, many algorithms had been proposed: carry look-ahead, carry-select, carry-skip, ling adders, etc [7-10]. The adder which was used in this paper makes a little difference with others. It effectively the effective propagation delay and so increases the frequency. It employs a combination of carry-skip and carry-select techniques. It was also used in [11]. Figure 4 shows the block diagram of this adder.

It consists of a ripple-carry adder blocks where each successive block is one bit longer than the block immediately below along with a carry-skip path jumping over each adder block. The delays in the ripple-carry adders and the carry-skip path are well balanced so that every carry propagates from the LSB to the MSB without waiting for the results from the other blocks.

If two or more bits of $x_i$, $y_i$ and $g_i$ are ‘1’ in the $i − th$ full-adder cell, carry $g_{i+1} = 1$ is generated and fed to the next cell. The cell never generates a carry if both $x_i$ and $y_i$ is ‘0’, regardless of the input $g_i$. If $g_{i+1} = 0$, either $x_i$ or $y_i$ is ‘0’ and the others is ‘1’, it will generate a carry if $c_j = 1$ comes up from the lower ripple-carry adder block $j−1$. For example, when $(x_3, x_4, x_5) = (1, 1, 1)$ and $(y_1, y_2) = (0, 0)$, block 1 does not generate carries $g_2, g_3$. However, if the carry $C_1 = 1$ reaches the block, the carry output $C_2$ immediately becomes ‘1’. This means that the carry $c_j$ can skip over the blocks one after another.
by pre-calculating a condition between $x_i$ and $y_i$ in each adder block $j$. The condition is defined by

$$P_j = \prod_i x_i \oplus y_i = 1,$$

where $\oplus$ is XOR.

By making the adder block size bigger toward the MSB side, the propagation time of $P_j$ and $C_j$ are equalized, and therefore the total delay time is minimized. Output $z_i$ initially holds a sum as if the block carry $C_j$ is ‘0’, and is inverted by the XOR gate if $C_j = 1$ comes up later.

5. Performance evaluation

The processor’s RTL model is structural and it was designed using VHDL [12-15] and RENOIR 2000.4 [16-17]. For each block in figure 1 and table 1, there is a corresponding symbol and an architectural body in RENOIR. All design units are synthesizable by Leonardo 2001[18]. Modelsim (version SE 5.5) is used to simulate and evaluate the architecture [19].

The design’s clock cycle was determined by calculating each element’s delay using the timing information in the SCL05u of ASIC at Leonardo 2001. Thus critical path is found and it’s delay time is inserted.

In the improved design with high-speed adder, the critical path delays occur at the execution stage. The longest delay inside the core is reduced from 32.8 ns to 18.797 ns. These values improve the operation frequency from 25.964 MHz to 37.81 MHZ.

So, the 45.6% improvement is reached by using the high-speed adder.

6. Conclusion

MIPS processors are widely used RISC processors in industry and research area. In this paper, a previously designed RISC processor which was constructed on basis of MIPS instruction set architecture is improved. The previous processor is briefly reviewed and a high-speed hybrid adder which employs both carry-skip and carry-select techniques, is designed. It was shown, that the maximum attainable frequency is increased from 25.964 MHz to 37.81 MHz.

Future work can be concentrated on the paths related with branch instructions which consists the critical path in current processor. Impleyng the sophisticated branch prediction methods will affectively increase the frequency and performance of the processor.

References: