Design of a Pipelined CPU including FPU based on SIMD Architecture

Luo, Yuxin

(出版者 / Publisher)
法政大学大学院情報科学研究科
(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 情報科学研究科編 / 法政大学大学院紀要. 情報科学研究科編
(巻 / Volume)
9
(開始ページ / Start Page)
21
(終了ページ / End Page)
26
(発行年 / Year)
2014-03
(URL)
https://doi.org/10.15002/00010514

Design of a Pipelined CPU including FPU based on SIMD Architecture

Yuxin Luo
Graduate School of Computer and Information Sciences
Hosei University
Tokyo, Japan
yuxin.luo.5h@stu.hosei.ac.jp

***Abstract*-Increasing demands for data processing capabilities require CPU not only with enormous integer computational capacities but also with sufficient speed to handle complicated floating point operation [4]. In this paper, according to the features of SIMD, we design a set instructions which includes integer and floating-point double data instructions. According to these set instructions, we design a pipelined CPU including the FPU based on the SIMD architecture to achieve high speed in handling integer and floating computing tasks. The CPU architecture consists of two SIMD processing modules which have their own registers and a shared memory. This allows highly and flexible computation of complicated integer and floating stream data tasks, which are difficult to deal with using a conventional CPU architecture. The goal of this research is to design a processor which can process the integer and floating-point based on SIMD architecture.**

Keywords—SIMD architecture, FPU, Pipeline

I. INTRODUCTION

As the core of the computer system, the traditional CPU is mainly used for general computing and control. With CPU being more and more widely used in communicating, image processing, transportation and many other fields, the requirement for the ability of data processing is becoming much higher, which requires the CPU with high speed and real-time processing ability for both integer and floating-point number[7].In this paper, we propose a pipelined CPU including FPU based on SIMD architecture to meet the requirements of high speed processing capability within different fields

In SIMD structure, all processing units execute the same instruction at any given clock, while each processing unit can operate on a different data element, such feature can greatly improve the speed of data processing.[3] In this paper, we design a set of SIMD instructions based on the characteristics of SIMD structure with which we can deal with integer arithmetic, floating point arithmetic, and the transformation of integers and floating point numbers, then we use Verilog HDL hardware language to realize this SIMD instruction set on CPU.

As shown in Fig. 1, the CPU we are talking about in this paper contains two FPU modules and two ALU modules, the floating point unit and integer unit have their own registers and they share the same data storage unit. Each floating point unit contains addition/subtraction, multiplication, division and square root .These four arithmetic modules which are pipeline designed to achieve the fast handling capability of floating point numbers. This SIMD-based CPU can perform parallel computing on integer and floating point numbers, this improves the speed of data processing on CPU, which can well satisfy the data processing requirements in some certain areas.

In the following sections, we will describe the design of the pipelined CPU including FPU based on SIMD.

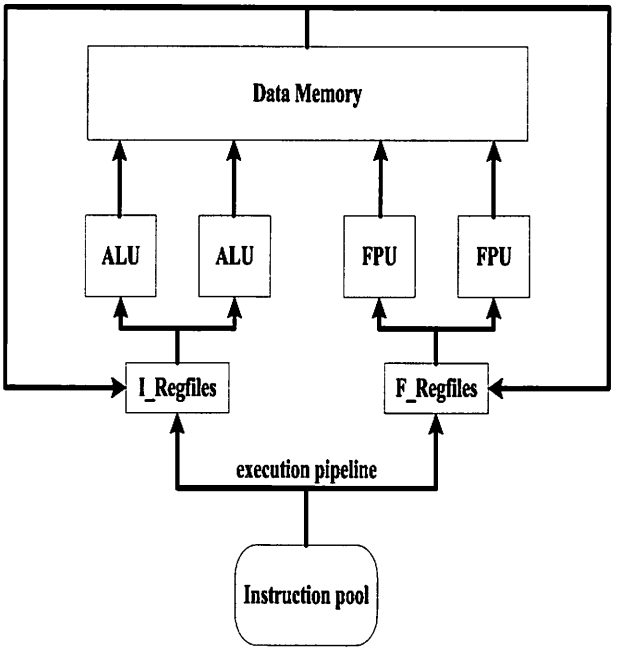


Fig. 1. SIMD CPU structure

II. PIPELINED CPU DESIGN

In this CPU, the integer instructions has 5 stages while 6 stages for floating-point instructions .The ITE(iteration) and ID stages are merged into one in the floating-point division instructions and the square root instructions, and fetching instruction operation is suspended in this stage . The operation of calculation and normalization of the Floating-point instructions will be divided into three stages: E1,E2,E3.As shown as the figure 2,all the floating-point instructions can be

describe by unified pipeline model: IF,ID,E1,E2,E3,WB,while the integer instructions can be describe by unified pipeline model : IF,ID,EXE,MEM,WB.

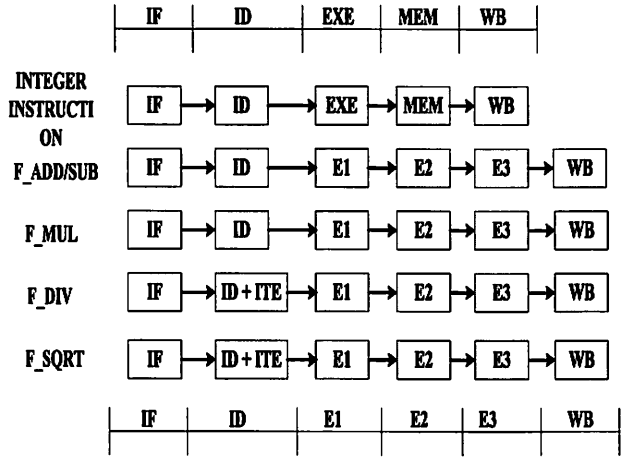


Fig. 2 Unified pipelined model of the CPU/FPU

In the pipelined CPU/FPU architecture, there are two important problems which must be solved: (1) Data Dependence; (2) Control Dependence. For solving these two problems, we use the below ways: (1) Internal Forwarding and suspend pipeline technology to solve data dependent; (2) Delayed Branch technology to deal with control hazard [1].

A. Internal forwarding for data dependence

Pipelined CPU can execute multiple instructions at one time. In this case, data dependence may happen among instructions which means the next instruction needs the result of the current instruction to execute while the current instruction has not completed yet. As shown as Fig. 3.

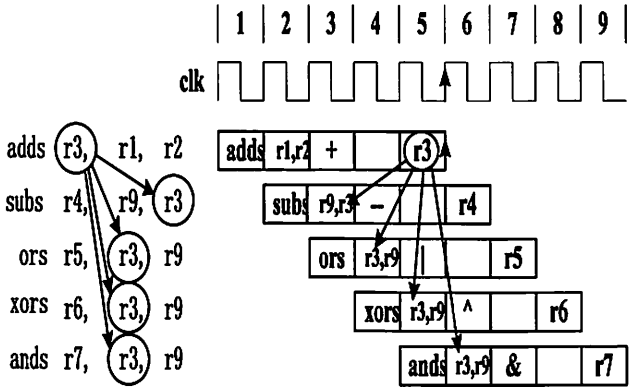


Fig. 3 Example of data dependence

The result of the first instruction(adds r3 , r1, r1) will be written into r3 after 5 cycles while the following instruction (subs , ors , ands , xors) needs to use the calculated values in r2 in ID stage. This means the following subs instruction cannot get wanted values in the ID stage.

To solve this problem, we adopt internal bypass way. As shown in the Fig. 4. The adds operation of the first instruction is finished at ALU of EXE stage. So we can directly deliver the result to the next instruction. That is to say, the result of ALU can be put forward to ID stage from EXE stage and MEM stage. Then the data dependent problem can be solved.

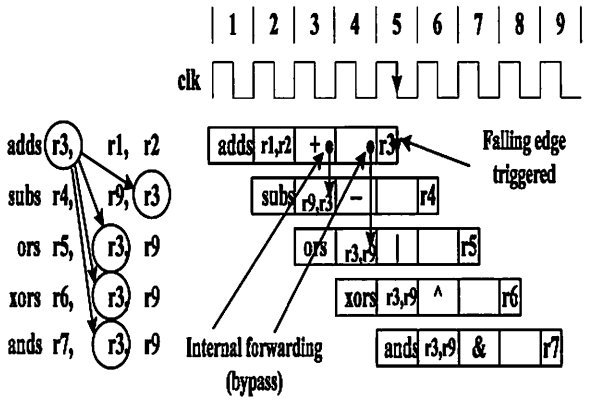


Fig. 4 Internal forwarding

B. Suspend pipeline for data dependence

The result of ALU can be pushed forward to ID stage from EXE stage or MEM stage. But the result of some instructions such as lws and lws2, would not appear until MEM stage finished. So the data only can be put forward to ID stage from MEM stage. If the next instruction of lws, lws2 is related to them, it needs to put data forward to ID stage from MEM stage and stall the pipeline for one cycle. As shown in the Fig. 5:

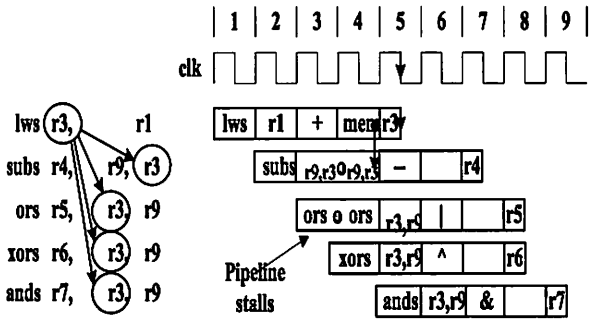
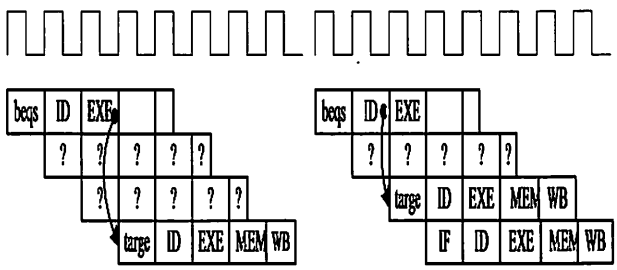


Fig. 5 Pipeline stall for data dependence.

C. Delayed transfers for control hazard

When Pipelined CPU performs transfer or jump instruction, there will appear control hazard problem. The transfer instruction or jump instruction has entered into the pipeline before the CPU turns into the destination addresses. As shown as an example in Figure 6:



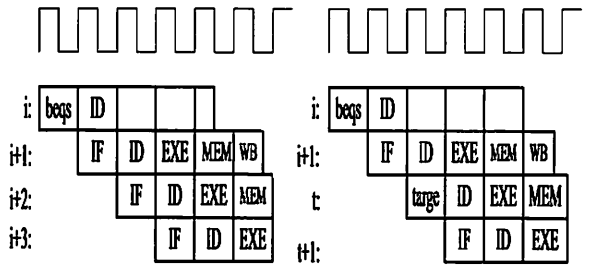
(a) : 2 slot (b): 1 slot

Fig. 6 Control hazard

Fig. 6 (a) shows how to define the transfer destination address and the conditions in the instruction beqs on EXE

level when two instructions following the beqs enter the pipeline, if the transferring destination address and the conditions are confirmed on ID stage, there will be only one subsequent instruction gets into the pipeline, as shown in Fig. 6(b).

We use a one cycle delay transfer technology to solve control hazard problem of the pipelined CPU.As shown as Fig. 7: Regardless of transferred or not, the instruction ((i + 1) -th) which follows the transfer instruction (i-th) will always be executed. This instruction set includes: jals, jrs, beqs and gts. It is to guarantee that jump instruction caused by every instruction can decide whether the transfer occurred and calculate the destination address on the ID stage.



(a): No transfer (b): Transfer

Fig. 7: One cycle delay transfer for pipelined CPU

jrs, jals are unconditional jump instruction which can be determined on the ID stage. beqs and gts are conditional jump instruction, it is necessary to compare if the data of two registers are equal on the ID stage with XOR gates.

III. FPU DESIGN

The FPU design is based on the IEEE 754 floating point Standard. The single-precision floating-point decimal (32 bits) is:

$$F = (-1)^s \times 1.f \times 2^{e-127}.[1] \qquad (1)$$

This paper contains the following floating-point instructions:

- I type:

flws, fsws,flws2,fsws2

- FR type:

fadds,fsubs,fmuls,fdivs,fsqrts,fi2fs,ff2is;fadds2,fsubs2,fmuls2,fdivs2,fsqrts2,fi2fs2,ff2is2;

FR type's instructions are used for floating-point arithmetic. When execute these instructions, there are 6 pipeline stages: IF, ID, E1, E2, E3, WB.

The FPU can execute the conversion between single precision floating-point value and integer value, add\subtract\multiply\divide and square root-operation of single precision floating-point value. Because the conversion between floating and integer is simple, there just need 2 stages when executing instructions of ff2is, fi2fs, ff2is2, fi2fs2. Other floating-point instructions execution process as shown as Fig. 8.

	fadds/fsub	fmuls	fdivs	fsqrts
ID			Newton-iteration	Newton-iteration
E1	exponent alignment	calculate partial product	calculate partial product	calculate partial product
E2	calculate	add partial product	add partial product	add partial product
E3	normalization	normalization	normalization	normalization

Fig. 8 FPU execution process

Instructions of fadds and fsubs have the same calculation structure. There is a select signal to judge whether execute fadds or fsubs. The fmuls instruction design and implement with Wallace Tree Algorithm. The fdivs instruction algorithm is same to the fmuls instruction, the main difference is the exponent computing. When execute fdivs construction, for example

$$a \div b = a \times 1 / b \qquad (2)$$

We use Newton-Raphson algorithm to calculate the value of 1/b, and then use the Wallace Tree algorithm to calculate the quotient. For the fsqrts instruction, we also use Newton-Raphson algorithm. It consists of two part operation: Iteration and square root calculation.

As shown as Fig. 9, integer and floating-point converter, floating-point adder, subtractor, multiplier, divider, square root device constitute the FPU. i_f is a select signal which judges the result of I2F and F2I.sel[0] decides the FADDER executes whether an addition or subtraction. sel[2:1] is a selection signal of a selector which judge the FPU output from the result of FAADR,FMUL,FDIV,FSQRT.

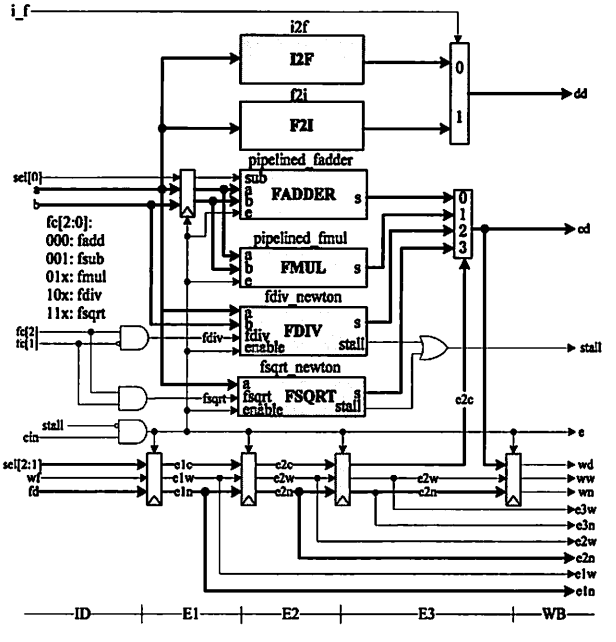


Fig. 8 Pipelined FPU

The result of FAADR, FMUL, FDIV, FSQRT must wait to the E3 stage, before this stage we not get the final result. There are 4 signals stall pipelined FPU.

- stall_fpu: Data dependence between floating-point instructions cause to stall pipeline;
- stall_flw: Data dependence between floating-point instructions and flws,flws2 cause to stall pipeline;
- stall_fsw: Data dependence between floating-point instructions and fsws,fsws2 cause to stall pipeline;
- stall_div_sqrt: fdivs, fdivs2, fsqrts, fsqrts2 wait Newton-Raphson algorithm to complete iteration operation.

IV. PIPELINED CPU/ FPU BASED ON SIMD ARCHITECTURE DESIGN

A. SIMD

SIMD is short for Single Instruction/Multiple Data, while a SIMD operation refers to a computing method that enables processing of multiple data with a single instruction. In contrast, the conventional sequential approach using one instruction to process each individual data is called scalar operations. As shown as Fig. 9, we can know the SIMD structure has the below features:

- Single Instruction: All processing units execute the same instruction at any given clock;
- Multiple Data: Each processing unit can operate on a different data element;
- Best suited for specialized problems characterized by a high degree of regularity, such as image/graphics processing;
- Synchronous and deterministic execution;

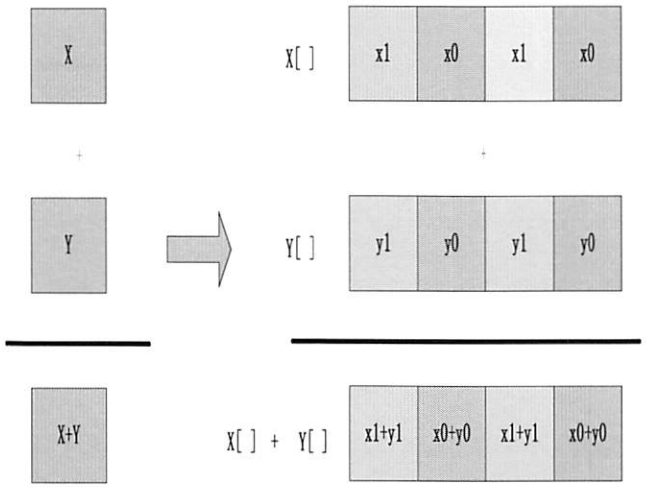


Fig. 9 an example of SIMD calculation

B. SIMD instructions

According to the features of SIMD, we design a set of single instruction double data instruction which include integer and floating-point instructions. As shown as Table 1, the instructions are double data. These are only double data instructions, there are also some effective single data instructions .For example, the instruction of ***movs*** which is a single data instruction, but execute ***movs*** instruction in the pipeline architecture, only need 2 stages.

TABLE I. SOME EXAMPLE OF INSTRUCTIONS

Instructions	Examples	Meanings
adds2	adds2r6, r4 ,r2	r6 = r4 + r2 r7 = r5 + r3
subs2	subs2 r6 , r4 ,r2	r6 = r4 - r2 r7 = r5 - r3
ands2	ands2 r6 , r4 ,r2	r6 = r4 & r2 r7 = r5 & r3
ors2	ors2 r6 , r4 ,r2	r6 = r4 | r2 r7 = r5 | r3
xors2	xors2 r6 , r4 ,r2	r6 = r4 ^ r2 r7 = r5 ^ r3
lws2	lws2 r4 , r2	r4 = memory[r2] r5 = memory[r3]
sws2	sws2 r4 , r2	memory[r2] = r4 memory[r3] = r5
fadds2	fadds2f6,f2 ,f4	f6 = f2 + f4 f7 = f3 + r5
fsubs2	fsubs2 f6, f2 ,f4	f6 = f2 - f4 f7 = f3 - r5
fmuls2	fmuls2 f6 ,f2 ,f4	f6 = f2 × f4 f7 = f3 × r5
fdivs2	fdivs2 f6,f2 ,f4	f6 = f2 ÷ f4 f7 = f3 ÷ f5
fsqrts2	fsqrts2 f6 ,f2	f6 = (f2)1/2 f7 = (f3)1/2
flws2	flws2 f6 , r2	f6 = memory[r2] f7 = memory[r3
fsws2	fsws2 f6 , r2	memory[r2] = f6 memory[r3] = f7
fi2fs2	fi2fs2 f6 , f2	f6 = i_2_f[f2] f7 = i_2_f[f3]
ff2is2	ff2is2 f6, f2	f6 = f_2_i[f2] f7 = f_2_i[f3]

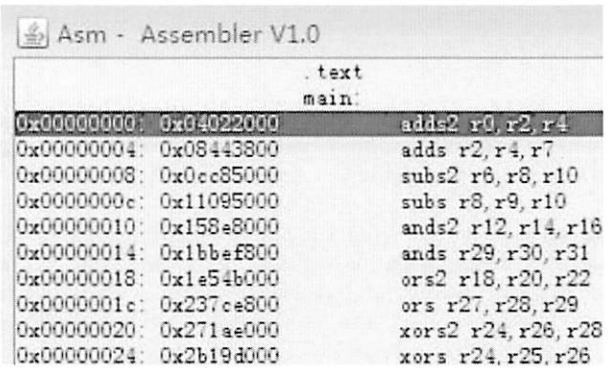


Fig. 10 Assembler for the SIMD instructions

As shown as Fig. 10, we design an assembler to translate these instructions to corresponding binary code for these SIMD instructions. The assembler can translate instructions to file.mif which is the original binary file of the instruction memory in Altera FPGA.

C. SIMD CPU architecture

According to the pipeline architecture and SIMD feature, we design a pipelined CPU including FPU based on SIMD architecture.

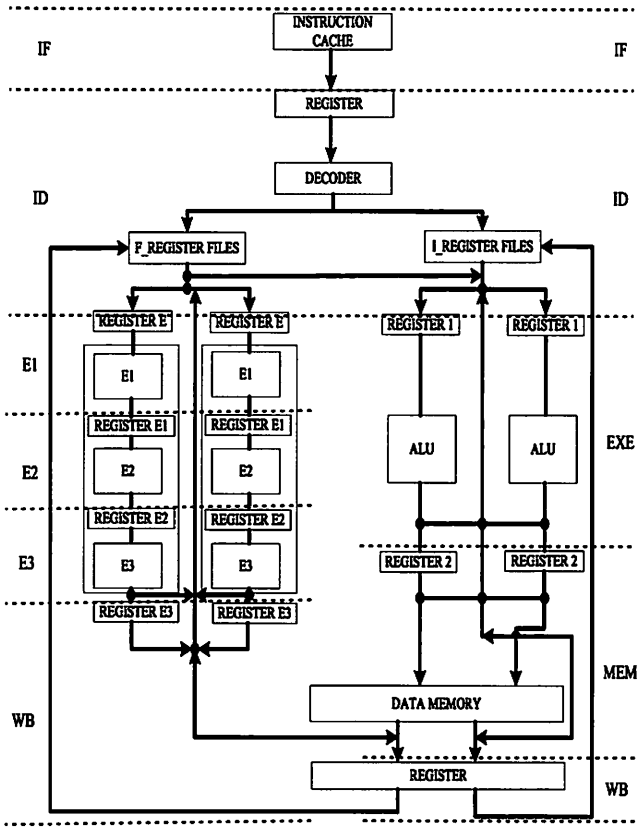


Fig. 11 CPU-FPU architecture

As shown as Fig. 11, the processor contains two model: floating-point calculation module and integer calculation module.

The floating-point calculation module is pipeline architecture and it has 6 stages which including two FPU. Each FPU can execute addition/subtraction, multiplication, division and square root. The integer calculation model also is pipeline architecture which includes 5 stages: IF, ID, EXE, MEM, WB. It has two ALUs which can execute integer and logic calculation.

The floating-point calculation module and the integer calculation module share the same storage unit while have their own registers. With the instructions of ff2is, ff2is2, fi2fs, fi2fs2, integer and floating can converse with each other. This CPU including FPU based on SIMD architecture can execute the set instructions of table 1. It can perform parallel computing on integer and floating-point data.

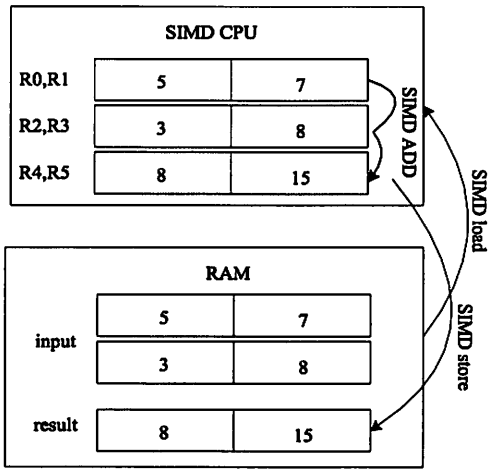


Fig. 12 Example of a SIMD calculation

As shown as Fig. 12, these is the calculation of *adds2* instruction. Two data streams execute add at the same cycle time. Store and load data are also the same way. These increase the calculation between data streams.

V. RESEARCH RESULTS

A. CPU containing only integer instructions test

The following program is used to test the CPU which is based on integer instructions.

Test codes:

```
0 : a8010000 ; %(00)   movs r1 ,4 %
1 : a8010000 ; %(00)   movs r2 ,8 %
2 : a8010000; %(00)    movs r3 ,16 %
3 : a8010000; %(00)    movs r4 ,20 %
4 : a8010000; %(00)    movs r5 ,24 %
5 : 38060000; %(00)    lws2 r6 ,r0 %
6 : 38480000; %(00)    lws2 r8 ,r2 %
7 : 38482000; %(00)    lws2 r10 ,r4 %
8 : 04084100; %(00)    adds2 r12 ,r6,r8 %
9 : 0406200; %(00)     adds2 r4 ,r10,r12 %
A : 62360000; %(00)    sws2 r5 ,r4 %
B : 27190000; %(00)    gts r5 ,r4, 16 %
```

Initial value of data memory:

```
0 : 00000000;  %(0) data[0] %
1 : 000000A3;  %(0) data[1] %
2 : 00000027;  %(0) data[2] %
3 : 00000079;  %(0) data[3] %
4 : 00000000;  %(0) data[4] %
5 : 00000143;  %(0) data[5] %
```

Test code meaning:

These codes means load the data memory's data[0],data[1],data[2],data[3],data[4],data[5], and add them, then store the result to data memory.

Test result:

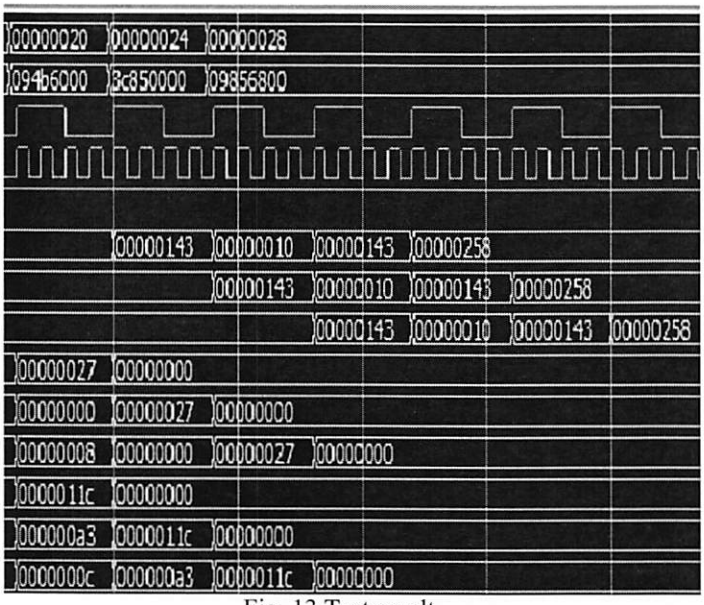


Fig. 13 Test result

As shown as Fig.13,the finally result 0x258, and add data[0] data[1],data[2],data[3],data[4],data[5] is also 0x258, that means the test result is right .

B. CPU including FPU based on SIMD test

Test result:

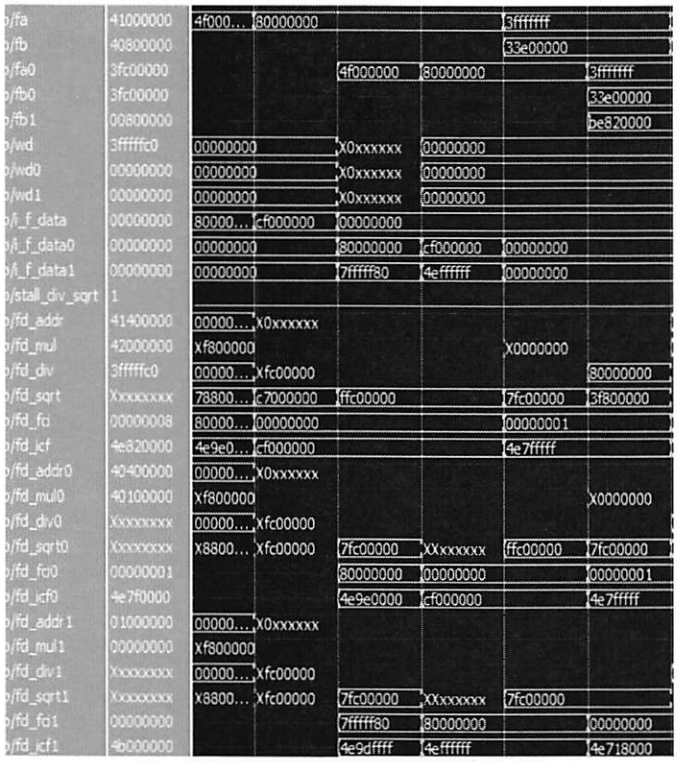


Fig. 14 CPU including FPU test result

As shown as Fig. 14, this is the result of the CPU including FPU. In this code, we test the double data instructions of floating-point instructions: fadds2, fsubs2, fdivs2, fsqrts2, ff2is2, ffi2fs2.

I. CONCLUSION

In this paper, we design a set of SIMD instructions which includes integer and floating-point instruction. According these instructions, we design a pipelined CPU including FPU based on SIMD architecture. The CPU has two SIMD processing modules which can perform optimized parallel processing for floating-point and integer processing. This SIMD CPU has two times speed than normal MIPS CPU.

In future work, we plan to continue to extend this work along several lines. From the application perspective, we look to add more efficient and concise instructions, for example, add the instruction which can execute four different data at the same time. We also want to use the TLP(thread level parallelism) way to design a fast CPU, and compare its performance with the SIMD CPU. Finally, we plan to use this CPU' set of instructions to build a 3D teapot model and implement it on the FPGA Board.

REFERENCES

[1] Li Ya Min,Computer Principles and Design in Verilog HDL,Tsinghua University press (2011.6).

[2] David A.Patterson, John L.Hennessy Computer Organization & Design,The Hardware / Software Interface .1997

[3] J. Tanabe, Y. Taniguchi, T. Miyamori, Y. Miyamoto, H. Takeda, M. Tarui,H. Nakayama, N. Takeda, K. Maeda and M. Matsui, "Visconti: Multi-VLIW image recognition processor based on configurable processor",Proceedings of the IEEE 2003 Custom Integrated Circuits Conference,pp. 185–188 (2003).

[4] D. Kim and D. Yeung, Design and evaluation of compiler algorithms for pre-execution. Proceedings of the Architectural support for programming.

[5] E. Gayles, T. Kellihr, R. Owens and M. Irwin, "The design of the MGAP-2: a micro-grained massively parallel array", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 8, 6, pp. 709–716 (2000).

[6] G.P. Biswas, P. Dutta, P. Krishna and I. Sengupta, "Cellular architecture for affine transforms on Raster images", IEE Proceedings

[7] C. Luk, S. Hong, H. Kim, Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. IEEE/ACM International Symposium on Microarchitecture, 2009.

[8] C. Zilles and G. Sohi, Execution-based prediction using speculative slices. International Symposium on Computer Architecture, 2001.

[9] T. Komuro, S. Kagami and M. Ishikawa, "A dynamically reconfigurable SIMD processor for a vision chip", IEEE Journal of Solid-State Circuits,39, 1, pp. 265–268 (2004).

[10] D. H. Woo, H. S. Lee, COMPASS: a programmable data prefetcher using idle GPU shaders. Proceedings of the Architectural support for programming languages and operating systems, March 13-17, 2010.

[11] S. Kyo, S. Okazaki and T. Arai, "An integrated memory array processor architecture for embedded image recognition systems", Proceedings of the 32nd International Symposium on Computer Architecture, pp. 134–145(2005).J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68-73.