dv-cpu-rv: CPU design of RISC-V

Devin

balddevin@outlook.com

2023/7/15

Content

1 Preface

The design of this CPU mainly refers to *Computer Organization and Design: The Hardware / Software Interface*: RISC-V Edition, David A.Patterson, John L. Hennessy.

The source code is also distributed to Github: devindang/dv-cpu-rv. Please feel free to submit issue or pull request.

2 Hardware Design

2.1 Basic Single Cycle Implementation

2.1.1 Deisng Diagram

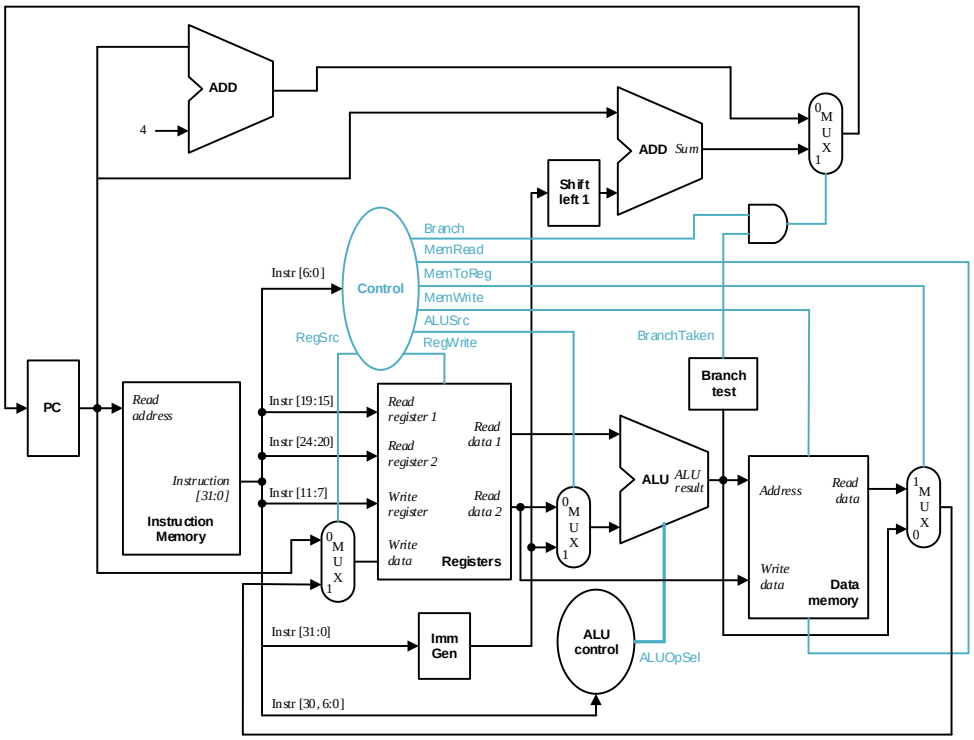


Figure 1.1 The Basic Single Cycle Implementation of CPU

The corresponding hash code is: a7b05c264b7f45e27a81ddc02184c6dcee29fdf9

The figure above shows the implementation of the basic single cycle cpu, in which the hinted lines are signals of control path, while the others are signals of data path.

2.1.2 The Building of Data Path

A data path is a unit used to operate on or hold data within a processor. In the ISC-V implementation, the datapath elements include the instruction and data memories, the register file, the ALU, and dders.

2.1.3 The Building of Control Path

2.2 Basic Pilelined Implementation

2.2.1 Design Diagram

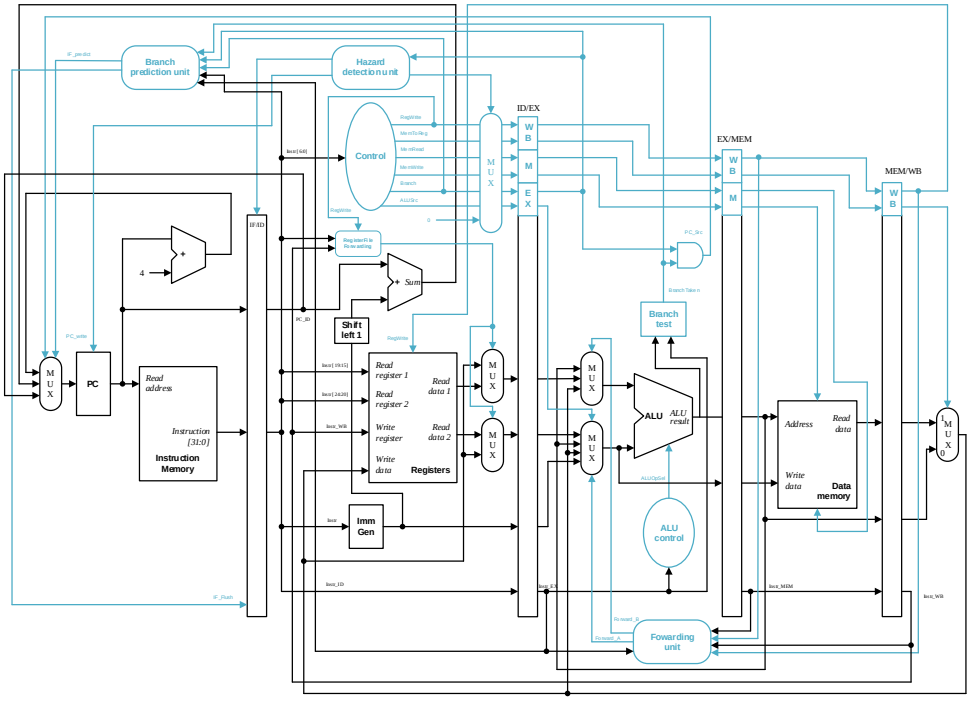


Figure 3.1 The basic pipelined implementation of CPU

The corresponding hash identifier is: 1d28ab2a485737b8bd90fa777fd550d5183b705c

The figure above shows the implementation of the basic pipelined cpu, in which the hinted lines are signals of control path, while the others are signals of data path. Compared to the single cycle implementation, additional units are required, they are:

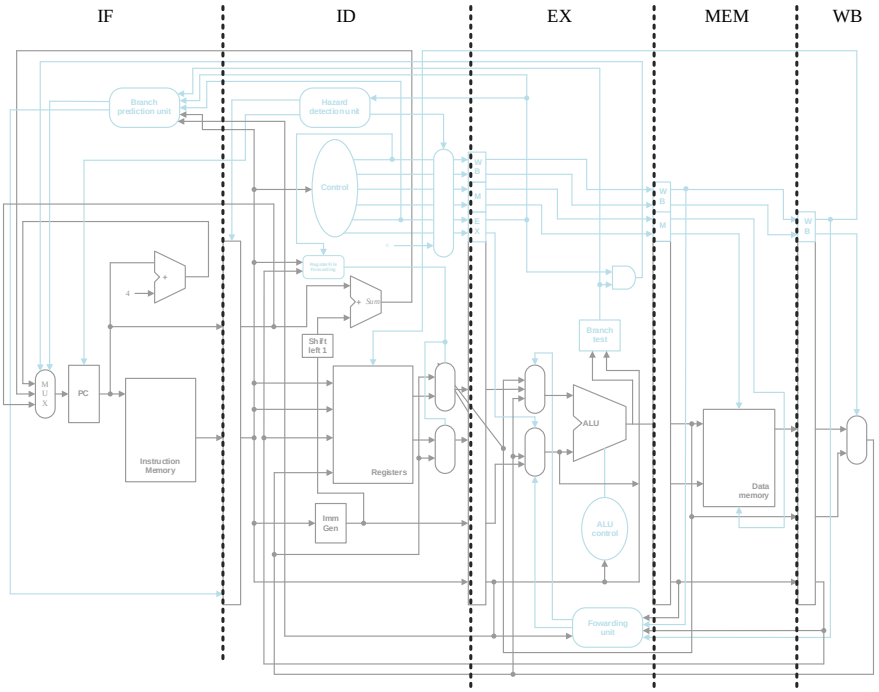
i) 4 registers for pipelining, named IF/ID, ID/EX, EX/MEM, MEM/WB, separately;

ii) Forwarding unit for dealing with data hazard introduced by pipelining;

iii) Hazard detection unit for stalling the CPU in special cases;

iv) Branch prediction unit for accelerating the CPU, which saves the operation cycles;

v) Forwarding unit for RegisterFile, which solve the read/write hazard of register.



2.2.2 Data Hazard: Forwarding or Bypass

Data hazards are obstacles to pipelined execution. The method to deal with this issue is adding a forwarding unit, which forwarding the data in the previous data flow, such as alu_result, or registered ones, to current execution cycle, instead of waiting for the last instruction to write back.

2.2.3 Control Hazard: Branch Prediction

3 Functional Description

3.1 Files and Directory Structure

The figure below shows the layout of the directories in the example system.

<home directory>	Local git directory.
｜ clean.pl	Perl script for cleaning temporary files.
├──core/	The implementation of CPU core.
｜ ├──bench/	Bench codes for CPU core.
｜ ├──rtl/	RTL codes.
｜ ├──sim/	VCS+Verdi simulation environment.
｜ └──vsim/	Modelsim simulation environment.

└──docs/ Related documentation.

3.2 Design of Modules in Details

3.2.1 Strobe Unit

Strobe unit is designed to implement instructions about Load and Store, they are:

- LB, LH, LW, LBU, LHU, SB, SH, SW in RV32I,
- LD, LWU, SD in RV64I.

The LD instruction loads a 64-bit value from memory into register rd for RV64I.

The LW instruction loads a 32-bit value from memory and sign-extends this to 64 bits before storing it in register rd for RV64I.

The LWU instruction, on the other hand, zero-extends the 32-bit value from memory for RV64I.

The LH and LHU instruction are defined analogously for 16-bit values, as are LB and LBU for 8-bit values. The SD, SW, SH, and SB instructions store 64-bit, 32-bit, 16-bit, and 8-bit values from the low bits of register rs2 to memory respectively.

The offset filed is in Bytes, for SD or LD instruction, the offset must be the multiplies of 8 (A doubleword is 8 bytes), for example,

```
ld x9, 240(x10) // Temporary reg x9 gets A[30]
lw x10, 240(x10) // Temporary reg x9 gets A[30][31:0]
```

The offset must be aligned.

4 Appendices

4.1 Appendix 1: Support of Instruction Set

The regularity of opcode:

The example of RISC-V pseudoinstructions can be found in *risc-v specification* v2.2 p109, from which we can deal with the assembly codes.

RV32I Base Instruction Set

Total: 47

Type	Order	Instruction	Description	Compatibility
R-type	1	ADD	Add.	YES
	2	SUB	Subtract.	YES
	3	SLL	Shift Left Logical.	YES
	4	SLT	Set Less Than.	YES

	5	SLTU	Set Less Than Unsigned.	YES
	6	XOR	Exclusive or.	YES
	7	SRL	Shift Right Logical.	YES
	8	SRA	Shift Right Arithmetic.	YES
	9	OR	Or.	YES
	10	AND	And.	YES
I-type	11	JALR	Jump And Link Register.	
	12	LB	Load Byte.	YES
	13	LH	Load Halfword.	YES
	14	LW	Load Word.	YES
	15	LBU	Load Byte Unsigned.	YES
	16	LHU	Load Halfword Unsigned.	YES
	17	ADDI	Add Immediate.	YES
	18	SLTI	Set Less Than Immediate.	
	19	SLTIU	Set Less Than Immediate Unsigned.	
	20	XORI	Exclusive Or Immediate.	
	21	ORI	Or Immediate.	
	22	ANDI	And Immediate.	
	23	SLLI	Shift Left Logic Immediate.	
	24	SRLI	Shift Right Logic Immediate.	
	25	SRAI	Shift Right Arithmeic Immediate.	
S-type	26	SB	Store Byte.	YES
	27	SH	Store Halfword.	YES
	28	SW	Store Word.	YES
B-type	29	BEQ	Branch if Equal.	
	30	BNE	Branch Not Equal.	
	31	BLT	Branch Less Than.	
	32	BGE	Branch Greater or Equal.	
	33	BLTU	Branch Less Than Unsigned.	
	34	BGEU	Branch Greater or Equal Unsigned.	
U-type	35	LUI	Load Upper Immediate.	YES
	36	AUIPC	Add Upper Immediate to PC.	
J-type	37	JAL	Jump And Link.	
other	38	FENCE		NO
	39	FENCE.I		NO
	40	ECALL		NO
	41	EBREAK		NO
	42	CSRRW		NO
	43	CSRRS		NO
	44	CSRRC		NO
	45	CSRRWI		NO
	46	CSRRSI		NO
	47	CSRRCI		NO

RV64I Base Instruction Set (in addition to RV32I)

Total: 15

Type	Order	Instruction	Description	Compatibility
R-type	1	ADDW	Add Word.	
	2	SUBW	Subtract Word.	
	3	SLLW	Shift Left Logical Word.	
	4	SRLW	Set Less Than Word.	
	5	SRAW	Set Less Than Unsigned Word.	
I-type	6	LWU	Load Word Unsigned.	YES
	7	LD	Load Doubleword.	YES
	8	SLLI	Shift Left Logic Immediate.	
	9	SRLI	Shift Right Logic Immediate.	
	10	SRAI	Shift Right Arithmetic Immediate.	
	11	ADDIW	Add Immediate Word.	
	12	SLLIW	Shift Left Logic Immediate Word.	
	13	SRLIW	Shift Right Logic Immediate Word.	
	14	SRAIW	Shift Right Arithmetic Immediate Word.	
S-type	15	SD	Store Doubleword.	YES

4.3 Appendix 2: Examples for Run

4.3.1 Example 1: Add and Store.

Date: 2023/7/23

Hash: a7b05c264b7f45e27a81ddc02184c6dcee29fdf9

Description: Given two numbers, add them and store into memory.

C code

```c
#include "stdio.h"
int main() {
          int a = 14;
          int b = 15;
          int c;
          c = a + b;
          return 0;
}
```

Assembly code

```
addi x2 x0 14;      //0//     00000000111000000000000100010011
addi x3 x0 15;      //1//     00000000111100000000000110010011
add  x1 x2 x3;      //2//     00000000000110001000000010110011
sd   x1 8(x2);      //3//     00000000000010001001101000100011
```

4.3.2 Example 2: Sum Less Than.

Date: 2023/7/29

Hash: 1d28ab2a485737b8bd90fa777fd550d5183b705c

Description: Given a non-zero natural number N, calculate the sum of natural numbers less than N.

C code

```c
#include "stdio.h"
int main() {
          int N = 10;
```

```
        int sum = 0;
        for(int i=1; i<N; i++){
                sum = sum+i;
        }
        return 0;
}
```

Assembly code

```
addi x1 x0 10;      //0//     00000000101000000000000010010011
addi x2 x0 1;       //4//     00000000000100000000000100010011
addi x3 x0 0;       //8//     00000000000000000000000110010011
add  x3 x2 x3;      //12//    00000000001100010000000110110011
addi x2 x2 1;       //16//    00000000000100010000000100010011
blt  x2 x1 A12;     //20//    11111110000100010100110011100011
sd   x3 8(x1);      //24//    00000000001100001011010000100011
```