Project 4 report

120090643 陈启旭

1. Big picture thoughts and ideas

   A CPU is a device that executes computer program instructions.

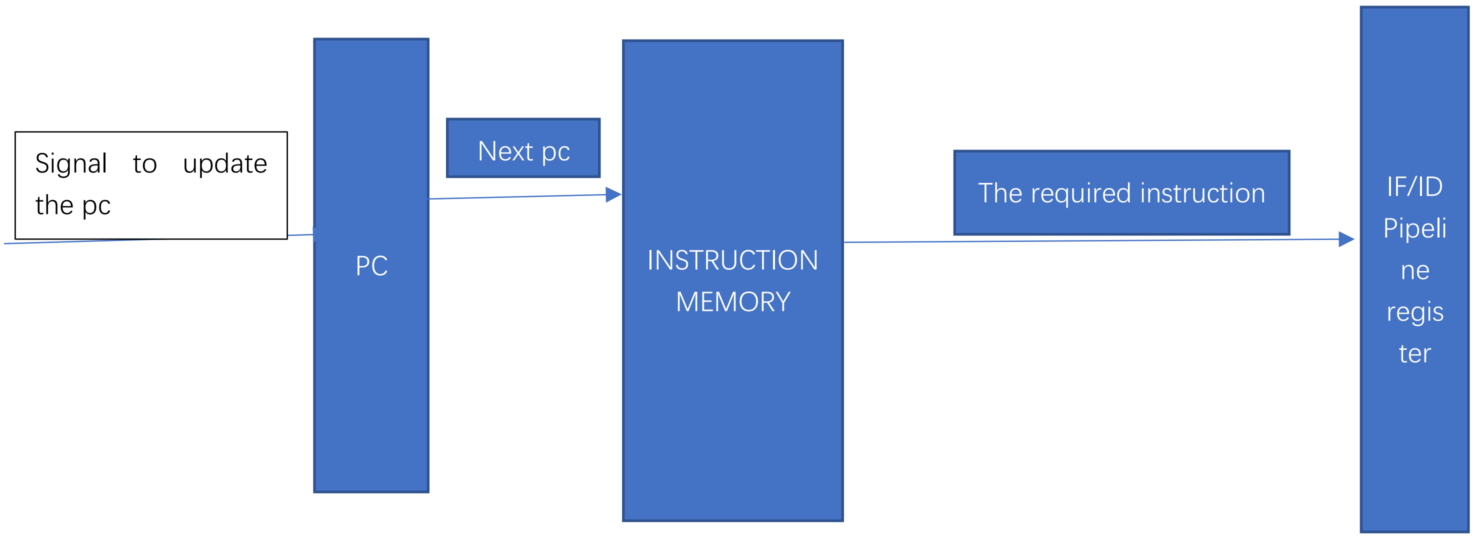
   A five-stage pipelined CPU add four pipeline registers to single cycle CPU, which enable the processor to execute one instruction in 5 cycles.

   The four pipeline registers divide the whole CPU into five parts which have different functions in each cycle:

   A. The CPU read one instruction from the instruction memory with the updated pc address in the first clock cycle.
   B. In the second cycle, the processor will split the instruction into many parts and use the registers and control unit to decode the MIPS instruction. The MIPS instruction's operation and function codes are delivered to the control unit, which recognizes the type of instruction.
   C. In the third cycle, ALU will handle arithmetic, logical, shifting, and conditional branch instructions.
   D. In the fourth cycle, data transfer instructions will fetch data from or store data to the data memory.
   E. In the fifth cycle, data will be written back to registers.

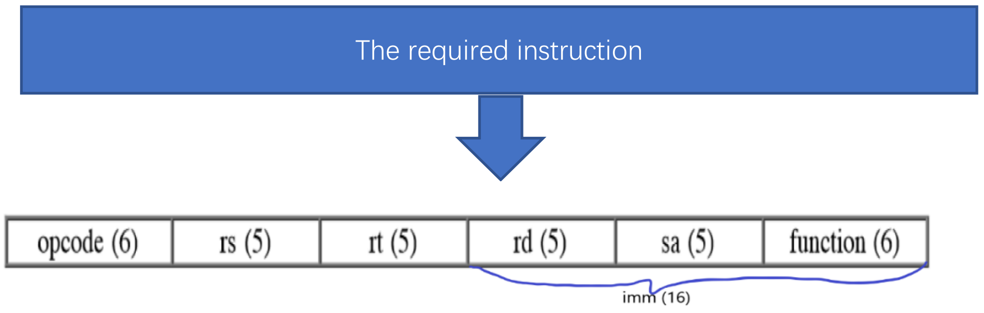
2. Data flow chart

   A. The IF stage

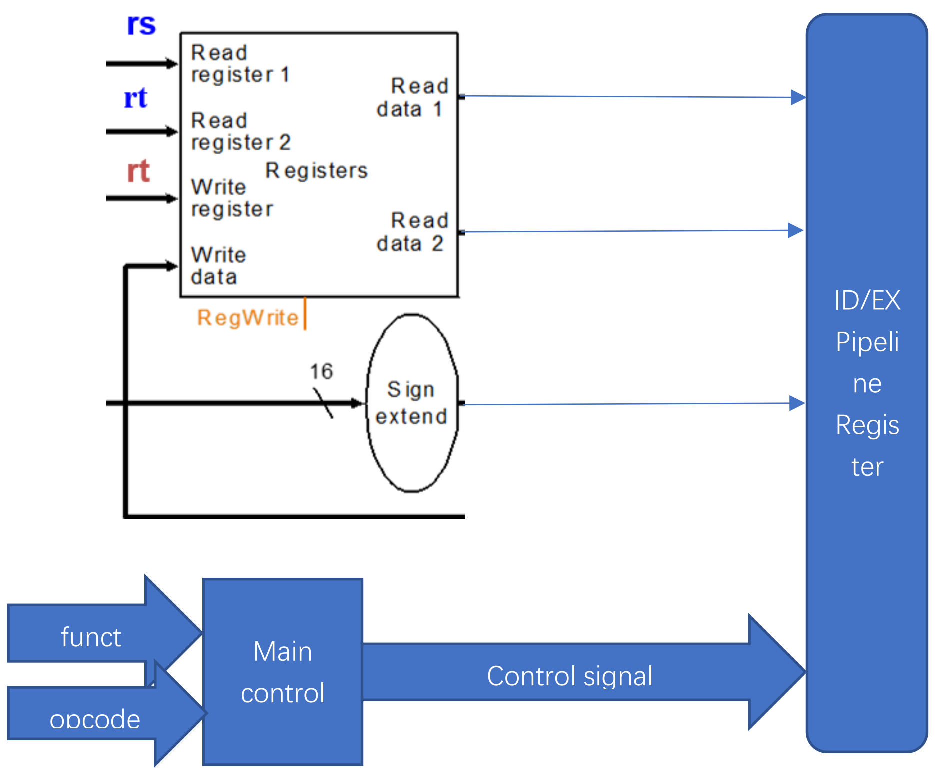


   B. The ID stage

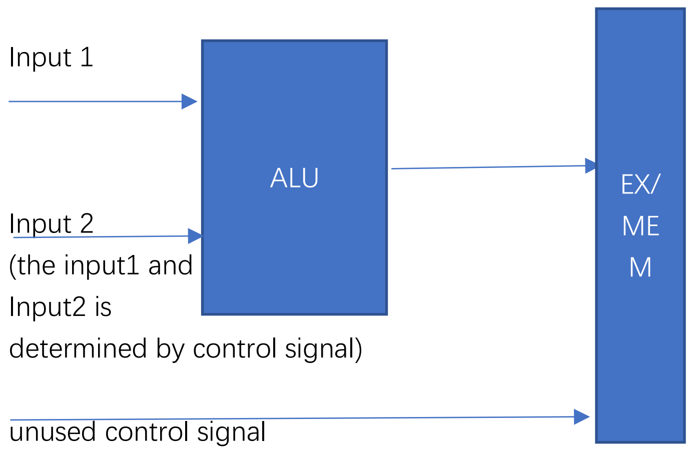
   Split the required instruction



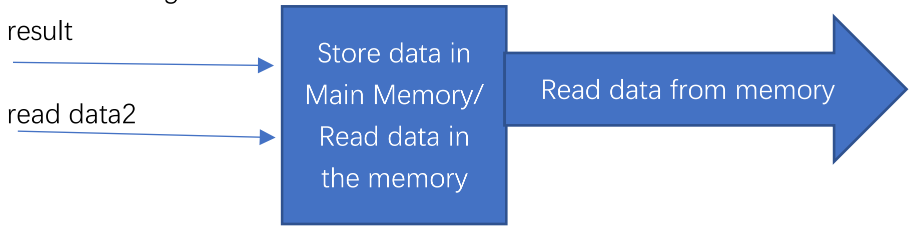
Fetch the value from the register file and convey the value to the pipeline register

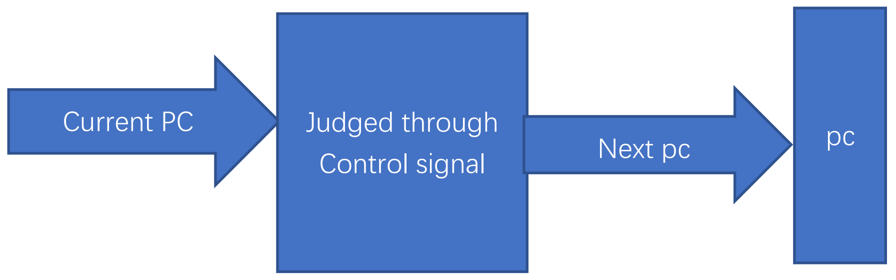


C. The EX stage

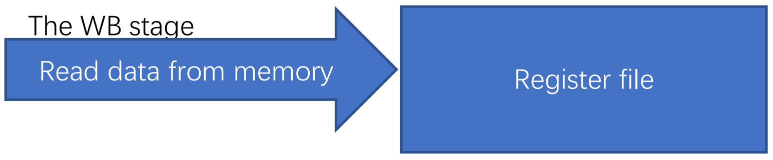


D. The MEM stage





E. The WB stage



3. High level implementation idea

3.1 PC module

A register to store the PC, initialized by 0: Receive the "next_pc" , update the current pc and output the pc to fetch the instructions.

Input: enable, clk, next_pc

Output: pc

3.2 Instruction RAM

A RAM which can store the instructions and output the required instructions.

Input clk; Input reset; Input enables; Input fetch address (pc);

Output: the required instruction

3.3 IFIDreg module

The first pipeline register, store the pc and instruction.

3.3 register file

A set of memory file which stores the value of 32 registers.

Input: readregister1(address of the first register); readregister2(address of the second register); writeregister (address of the target register to write in);

Regwrite (whether to write in the register)

Output: readdata1, readdata2(value stored in the first and second registers)

3.4 main control

Read in the opcode and funct and generate the control signal.

Input: opcode, funct

Output: A. **regdst**: judge which data to put in the write register of the register file.

If regdst equals to 10, the register file will write in the 31th register.

If regdst equals to 00, the register file will write in the rt register. (I type instruction)

If regdst equals to 01, the register file will write in the rd register (R type instruction)

B. **regwrite:**

If regwrite equals to 0, the register will not write in data.

If regwrite equals to 1, the register will write in data.

C. **alusrc**

If alusrc equals to 0, the second input to the ALU will come from the readdata2 from the register file.

If alusrc equals to 1, the second input to the ALU will come from the extended Immediate number.

D. **aluctr**

Aluctr will be inputted into the ALU to decide which kind of implementation will be implemented for the alu

For example:

If aluctr equals to 0010, the ALU will output the sum of the two input operands.

E. pcsrc

Decide which address will be used to update the pc.

If pcsrc and the zero are both 1, the pc will update to pc+4+imm<<2.

Else, the pc will update to pc+4

F. memread

if memread equals to 1, then read the content of the main memory

G. memwrite

if memwrite equals to 1, then write in the content of the main memory

H. Memtoreg

Decide which data will be written back to the register file.

If Memtoreg equals to 00, result of the ALU will be written back to the register file.

If Memtoreg equals to 01, the fetched value of the main memory will be written back.

If Memtoreg equals to 10, then pc+4 will be written back. (for jal operation)

I. Jump

If equals to 1, the pc will update to the jump address

J. Immctr

If equals to 1, zero extend the immediate; else, sign extend.

K. Jumpreg

If equals to 1, the pc will update to the register value (for the jr operation)

L. Pcbne

For bne and beq, if bne, the signal is assigned to 0, else,1.

3.5 imm

input: 16 bits immediate number, immctr signal;

output: 32 bits extended immediate number

3.6 IDEX reg

Stores the two readdata from register file, the current instruction， immediate number and all the control signal.

3.7 ALU: showed in the last project

3.8 EXMEMreg：

Pipeline register

3.9 MEMWBreg

Pipeline register

3.10 Instruction RAM

Input: clock, enable

Fetch_address

Output: If enable equals to 1, the module will output the instruction in Fetch_address.

3.11 Main Memory

Input: clock, enable, writedata

Fetch_address, memread, memwrite

Output: DATA

If memread equals to 1, the module will output the data in the Fetch_address.

If memwrite equals to 1, the memory will change the value stored in the Fetch_address to writedata.

4. implementation detail

4.1 How to distinguish beq and bne:

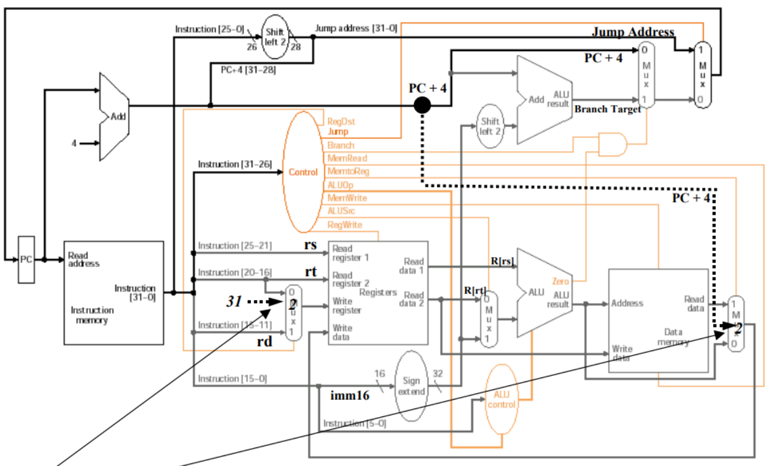
The program will generate a pcbne for every implementation.
If the instruction is beq, the value of pcbne will be assigned to 1
Else, the value will be assigned to 0.
To support the branch instruction, I designed the following process to select the updated PC
There is a wire called branch

```
assign branch = pcsrc&(~pcbne^MEMzero);
```

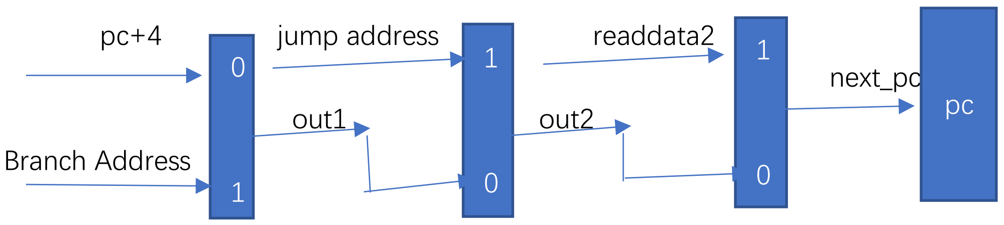
The program will use branch to select the pc

4.2 How to support j, jr and jal

The program has jumpreg and jump to support the instruction of j, jr and jal.
(These signals have been explained in the third part)
To support the jal function, the datapath is modified as follows:(pipelines omitted)



1. Expand the multiplexor controlled by RegDst to include the value 31 as a new input 2.
2. Expand the multiplexor controlled by MemtoReg to have PC+4 as new input 2.



pc+4
0
Branch Address
1
jump address
1
out1
0
readdata2
1
out2
0
next_pc
pc