THE CNINESE UNIVERSITY OF HONG KONG, SHENZHEN

CSC3050

Computer Architecture

Report for Project 4

Author:
Li Jingyu 李璟瑜

Student Number:
118010141

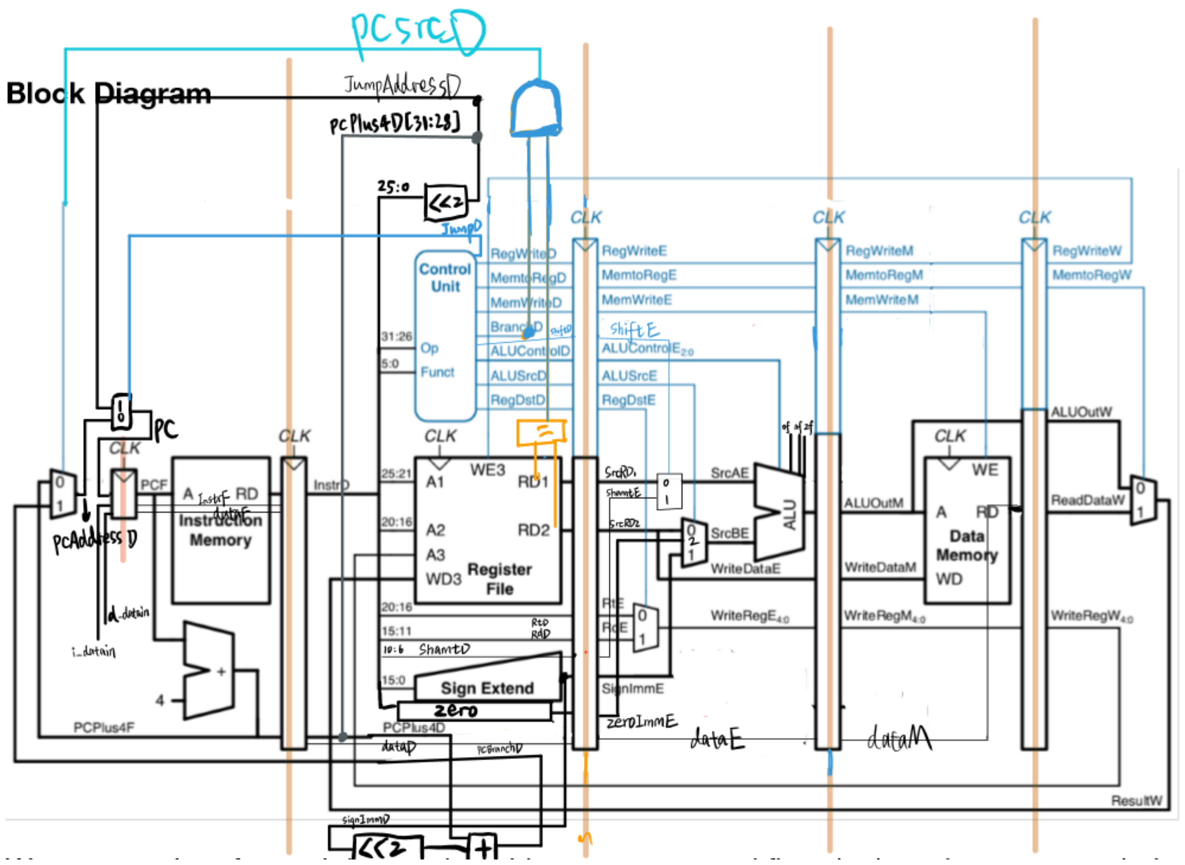
May 4, 2020

Contents

1 Block Diagram and Data path

Project 4 requires us to design a whole (pipeline) CPU module, which includes clock, instruction memory (use i_datain to represent), registers (control, store values), ALU (designed in project 3), data memory (use d_datain to represent) and control unit (determine the operation of different units / multiplexers). The CPU is divided into 5 pipeline stages, which are executed simultaneously. Such design improves the speedup (throughput).

About Block Diagram:



About Data path:

IF: Instruction Fetch. Receive i_datain and d_datain, upgrade pc according to branch and jump control (or +4). Transfer instruction and data to ID;

ID: Instruction Decode. Decode the 32-bit instruction; get the control wire and many values that will be used in the following stages. Get the values from register file (general registers). Extend the 16-bit immediate (Zero or Sign Extension). Calculate the branch & jump address, upgrade pc.

EX: Select the value according to the control wire. Use ALU to calculate the result.

MEM: Write the data to memory (sw) or fetch data from data memory (lw).

WB: Write the ALU result or loaded data to the general registers.

(I write detailed comments in the CPU.v code for each stage, about the registers used and input/output)

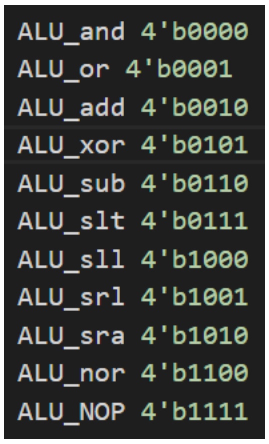
(I also write an additional file called CPU_para.v to define the parameter)

(Special register: DataX: Used to transfer the d_datain.)

About Task:(I write two tasks to finish the control unit in ID and ALU in EX)

Control_Unit: To decode. Receive opcode and funct from the instruction and output the control wire. (For example: RegWrite represents whether the value in WB will be written to the general registers; Shift: SrcAE will use shift value instruct[10:6] or RD1, general register value; Jump: decide whether to use the jump address ;ALUControl: Decide what alu operation will execute in the ALU task).

ALU: To calculate. Receive two values from EX stage and ALUControl wire, do the corresponding operations (add, sub, shift…), and output the ALU result.



```
ALU_and 4'b0000
ALU_or 4'b0001
ALU_add 4'b0010
ALU_xor 4'b0101
ALU_sub 4'b0110
ALU_slt 4'b0111
ALU_sll 4'b1000
ALU_srl 4'b1001
ALU_sra 4'b1010
ALU_nor 4'b1100
ALU_NOP 4'b1111
```

Flags like overflow, zero, and negative will also be outputted. They evaluate the state of the ALU result.

About Hazards:

In this project, I handle the branch hazard, which reduced two pipeline stages. In the second stage (ID), CPU compares the value of RD1 and RD2 and “AND” with the BranchD control, then send the PCSrcD control wire back to the first stage to select the pc. Normally, the branch control wire and result will be sent back in the fourth stage (MEM) and zero flag will be used. Here CPU just simplifies this process.

2 Results Display

This part shows the results of the CPU execution.

Items displayed: PC (IF), instruction code (IF), SrcA & SrcB (ALU part, EX), ALUOUT (MEM), Result (WB), data_in, values of the general registers 0-7 and $ra.

Names of the instruction are also displayed.

At first, $gr0-gr6 are set as 0, 1, 2, 3, 4, 5, -2 and they will not change during the execution. $ra is set as dd. It will be changed by jal instruction.

For the logical and arithmetic instructions, all the values are written in $gr7, so you can just check the value of $gr7 for all the ALU operations.

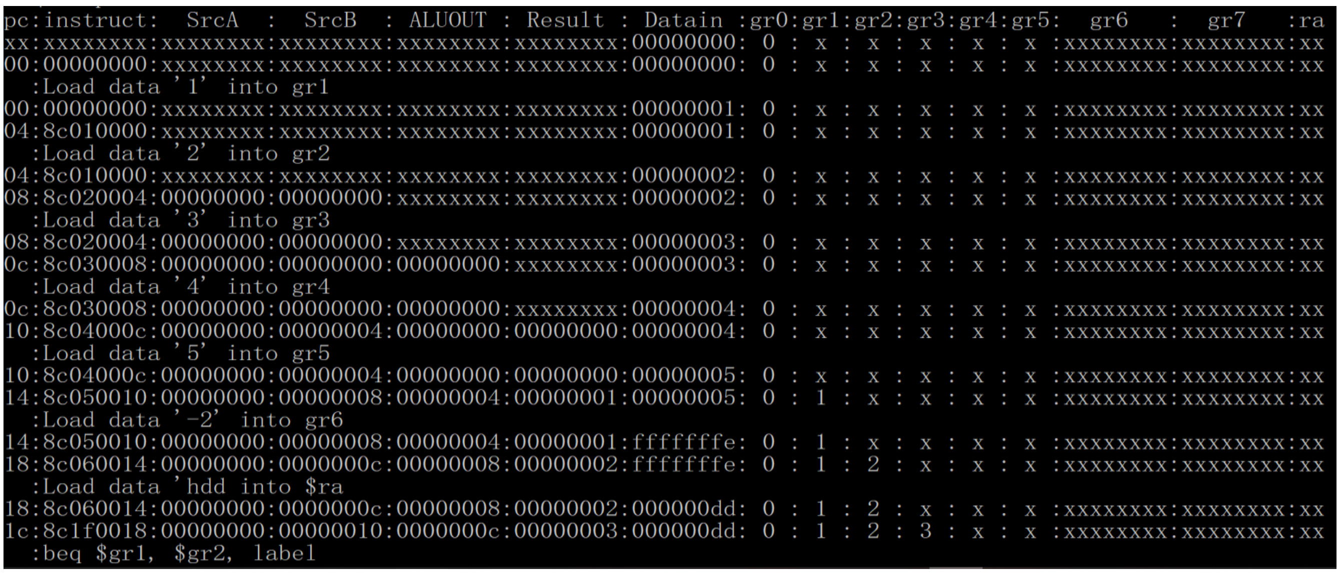
For the jump and branch instructions: j, jr, jal, beq, bne. Value of the pc will be changed two cycles later. (In ID stage, the values will be passed back to IF). Please check pc for these instructions.

Except for load instruction, other comments correspond to the pc in the next line. For example, add $gr7, $gr1, $gr2 → pc=24 (down). (However, for load the comment position is incorrect: first load should be at pc=4)

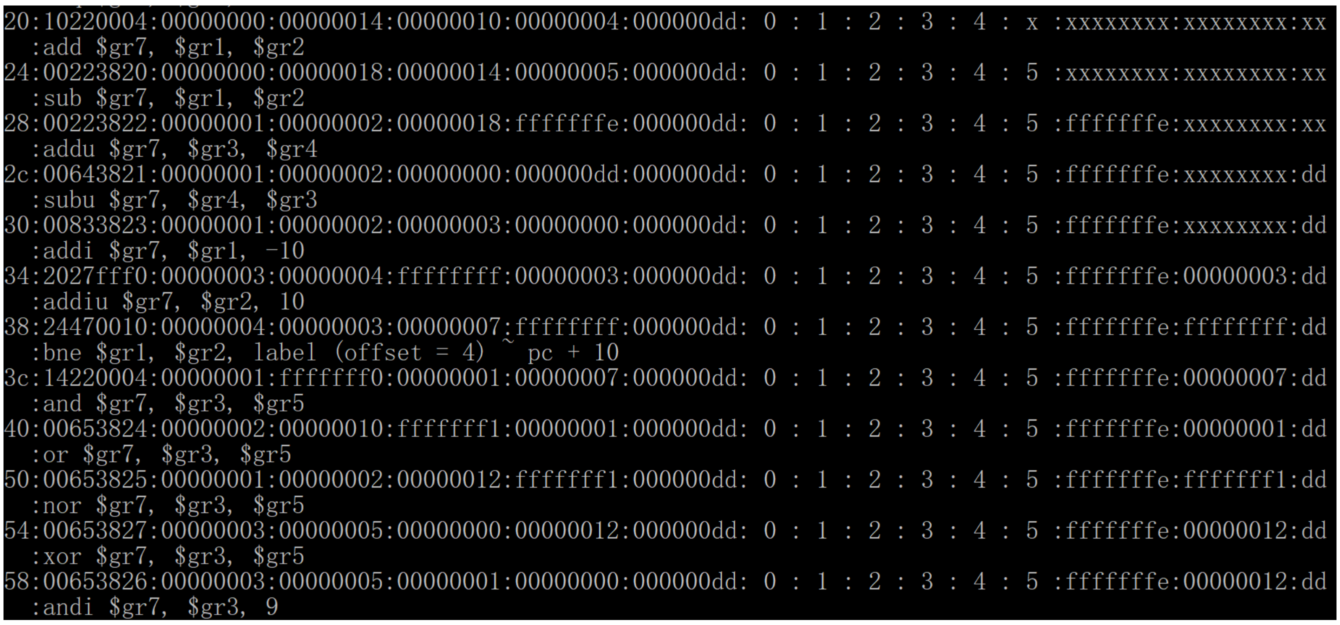
The results will be written to the corresponding registers after 5 clock cycles (pc+10).

Detailed instruction code and meaning can also be seen in test_CPU.v file.

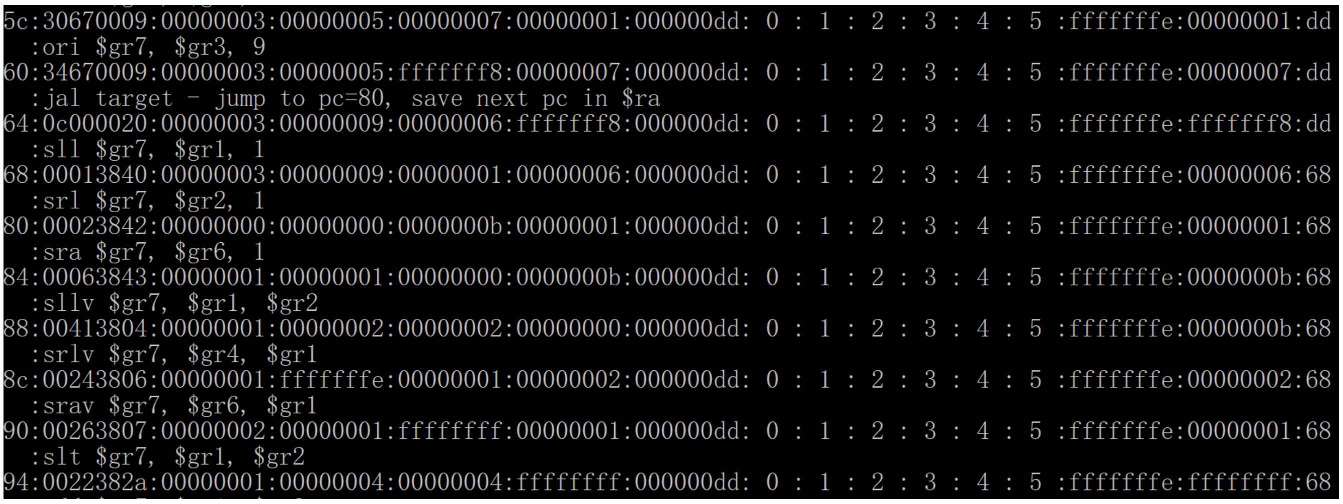
(Maybe it is a little bit difficult to check the results, so I have also uploaded the pictures in the zip file.



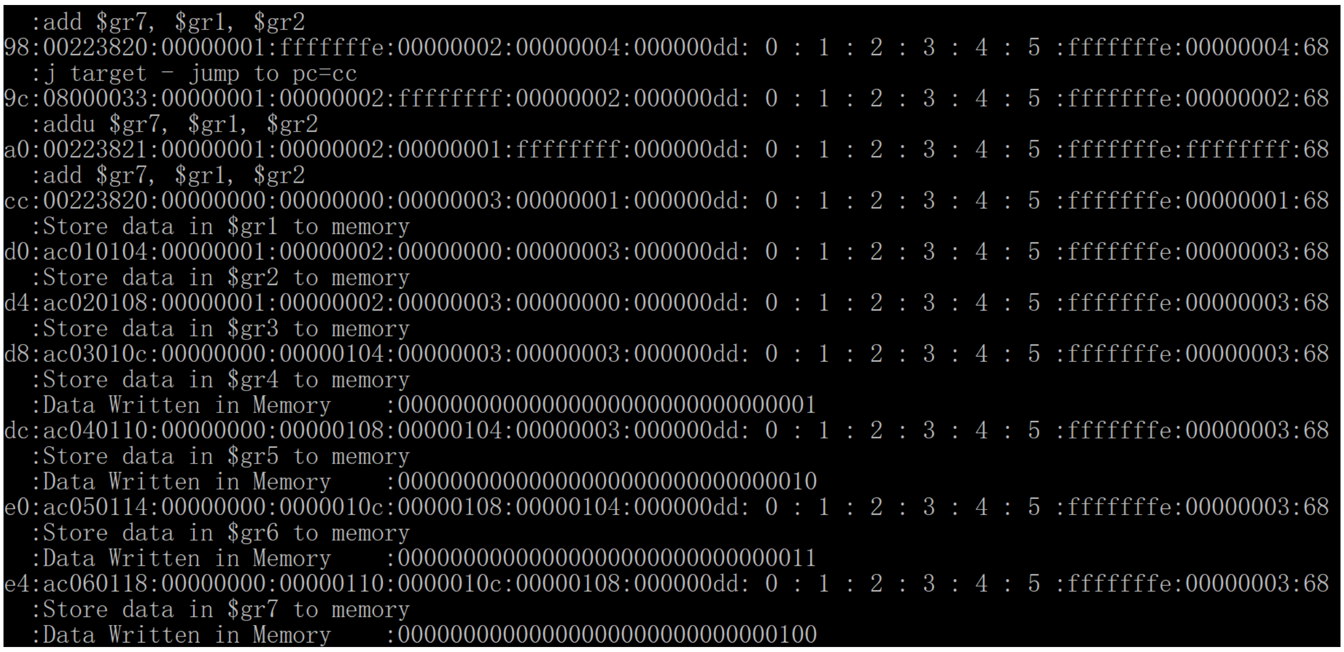
```
pc:instruct:  SrcA   :  SrcB   : ALUOUT  : Result  : Datain  :gr0:gr1:gr2:gr3:gr4:gr5:   gr6    :   gr7    :ra
xx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:00000000: 0 : x : x : x : x : x :xxxxxxxx:xxxxxxxx:xx
00:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:00000000: 0 : x : x : x : x : x :xxxxxxxx:xxxxxxxx:xx
  :Load data '1' into gr1
00:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:00000001: 0 : x : x : x : x : x :xxxxxxxx:xxxxxxxx:xx
04:8c010000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:00000001: 0 : x : x : x : x : x :xxxxxxxx:xxxxxxxx:xx
  :Load data '2' into gr2
04:8c010000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:00000002: 0 : x : x : x : x : x :xxxxxxxx:xxxxxxxx:xx
08:8c020004:00000000:00000000:xxxxxxxx:xxxxxxxx:00000002: 0 : x : x : x : x : x :xxxxxxxx:xxxxxxxx:xx
  :Load data '3' into gr3
08:8c020004:00000000:00000000:xxxxxxxx:xxxxxxxx:00000003: 0 : x : x : x : x : x :xxxxxxxx:xxxxxxxx:xx
0c:8c030008:00000000:00000000:00000000:xxxxxxxx:00000003: 0 : x : x : x : x : x :xxxxxxxx:xxxxxxxx:xx
  :Load data '4' into gr4
0c:8c030008:00000000:00000000:00000000:xxxxxxxx:00000004: 0 : x : x : x : x : x :xxxxxxxx:xxxxxxxx:xx
10:8c04000c:00000000:00000004:00000000:00000000:00000004: 0 : x : x : x : x : x :xxxxxxxx:xxxxxxxx:xx
  :Load data '5' into gr5
10:8c04000c:00000000:00000004:00000000:00000000:00000005: 0 : x : x : x : x : x :xxxxxxxx:xxxxxxxx:xx
14:8c050010:00000000:00000008:00000004:00000001:00000005: 0 : 1 : x : x : x : x :xxxxxxxx:xxxxxxxx:xx
  :Load data '-2' into gr6
14:8c050010:00000000:00000008:00000004:00000001:fffffffe: 0 : 1 : x : x : x : x :xxxxxxxx:xxxxxxxx:xx
18:8c060014:00000000:0000000c:00000008:00000002:fffffffe: 0 : 1 : 2 : x : x : x :xxxxxxxx:xxxxxxxx:xx
  :Load data 'hdd into $ra
18:8c060014:00000000:0000000c:00000008:00000002:000000dd: 0 : 1 : 2 : x : x : x :xxxxxxxx:xxxxxxxx:xx
1c:8c1f0018:00000000:00000010:0000000c:00000003:000000dd: 0 : 1 : 2 : 3 : x : x :xxxxxxxx:xxxxxxxx:xx
  :beq $gr1, $gr2, label
```



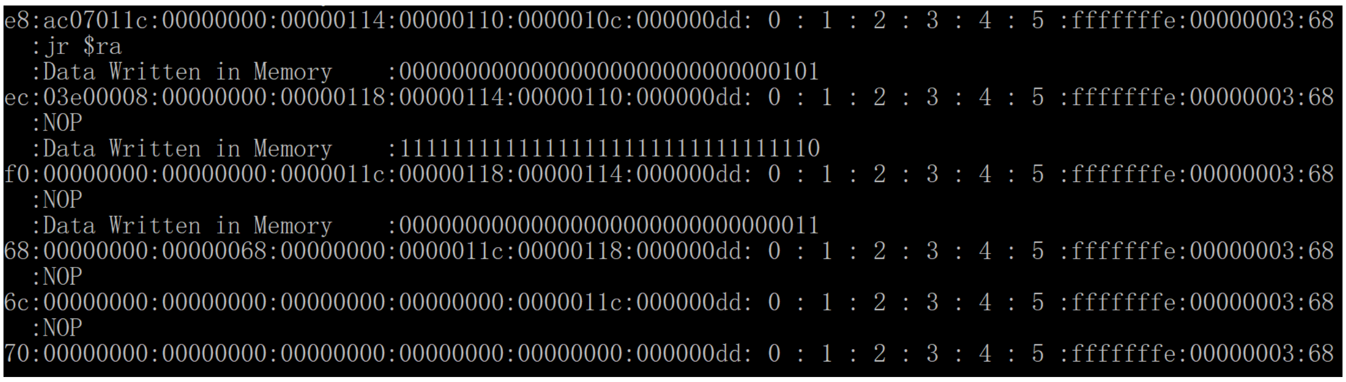
```
20:10220004:00000000:00000014:00000010:00000004:000000dd: 0 : 1 : 2 : 3 : 4 : x :xxxxxxxx:xxxxxxxx:xx
  :add $gr7, $gr1, $gr2
24:00223820:00000000:00000018:00000014:00000005:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :xxxxxxxx:xxxxxxxx:xx
  :sub $gr7, $gr1, $gr2
28:00223822:00000001:00000002:00000018:fffffffe:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:xxxxxxxx:xx
  :addu $gr7, $gr3, $gr4
2c:00643821:00000001:00000002:00000000:000000dd:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:xxxxxxxx:dd
  :subu $gr7, $gr4, $gr3
30:00833823:00000001:00000002:00000003:00000000:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:xxxxxxxx:dd
  :addi $gr7, $gr1, -10
34:2027fff0:00000003:00000004:ffffffff:00000003:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000003:dd
  :addiu $gr7, $gr2, 10
38:24470010:00000004:00000003:00000007:ffffffff:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:ffffffff:dd
  :bne $gr1, $gr2, label (offset = 4) ~ pc + 10
3c:14220004:00000001:fffffff0:00000001:00000007:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000007:dd
  :and $gr7, $gr3, $gr5
40:00653824:00000002:00000010:fffffff1:00000001:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000001:dd
  :or $gr7, $gr3, $gr5
50:00653825:00000001:00000002:00000012:fffffff1:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:fffffff1:dd
  :nor $gr7, $gr3, $gr5
54:00653827:00000003:00000005:00000000:00000012:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000012:dd
  :xor $gr7, $gr3, $gr5
58:00653826:00000003:00000005:00000001:00000000:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000012:dd
  :andi $gr7, $gr3, 9
```



```
5c:30670009:00000003:00000005:00000007:00000001:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000001:dd
  :ori $gr7, $gr3, 9
60:34670009:00000003:00000005:fffffff8:00000007:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000007:dd
  :jal target - jump to pc=80, save next pc in $ra
64:0c000020:00000003:00000009:00000006:fffffff8:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:fffffff8:dd
  :sll $gr7, $gr1, 1
68:00013840:00000003:00000009:00000001:00000006:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000006:68
  :srl $gr7, $gr2, 1
80:00023842:00000000:00000000:0000000b:00000001:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000001:68
  :sra $gr7, $gr6, 1
84:00063843:00000001:00000001:00000000:0000000b:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:0000000b:68
  :sllv $gr7, $gr1, $gr2
88:00413804:00000001:00000002:00000002:00000000:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:0000000b:68
  :srlv $gr7, $gr4, $gr1
8c:00243806:00000001:fffffffe:00000001:00000002:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000002:68
  :srav $gr7, $gr6, $gr1
90:00263807:00000002:00000001:ffffffff:00000001:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000001:68
  :slt $gr7, $gr1, $gr2
94:0022382a:00000001:00000004:00000004:ffffffff:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:ffffffff:68
```



```
  :add $gr7, $gr1, $gr2
98:00223820:00000001:fffffffe:00000002:00000004:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000004:68
  :j target - jump to pc=cc
9c:08000033:00000001:00000002:ffffffff:00000002:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000002:68
  :addu $gr7, $gr1, $gr2
a0:00223821:00000001:00000002:00000001:ffffffff:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:ffffffff:68
  :add $gr7, $gr1, $gr2
cc:00223820:00000000:00000000:00000003:00000001:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000001:68
  :Store data in $gr1 to memory
d0:ac010104:00000001:00000002:00000000:00000003:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000003:68
  :Store data in $gr2 to memory
d4:ac020108:00000001:00000002:00000003:00000000:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000003:68
  :Store data in $gr3 to memory
d8:ac03010c:00000000:00000104:00000003:00000003:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000003:68
  :Store data in $gr4 to memory
  :Data Written in Memory    :00000000000000000000000000000001
dc:ac040110:00000000:00000108:00000104:00000003:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000003:68
  :Store data in $gr5 to memory
  :Data Written in Memory    :00000000000000000000000000000010
e0:ac050114:00000000:0000010c:00000108:00000104:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000003:68
  :Store data in $gr6 to memory
  :Data Written in Memory    :00000000000000000000000000000011
e4:ac060118:00000000:00000110:0000010c:00000108:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000003:68
  :Store data in $gr7 to memory
  :Data Written in Memory    :00000000000000000000000000000100
```



```
e8:ac07011c:00000000:00000114:00000110:0000010c:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000003:68
  :jr $ra
  :Data Written in Memory    :00000000000000000000000000000101
ec:03e00008:00000000:00000118:00000114:00000110:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000003:68
  :NOP
  :Data Written in Memory    :11111111111111111111111111111110
f0:00000000:00000000:0000011c:00000118:00000114:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000003:68
  :NOP
  :Data Written in Memory    :00000000000000000000000000000011
68:00000000:00000068:00000000:0000011c:00000118:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000003:68
  :NOP
6c:00000000:00000000:00000000:00000000:0000011c:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000003:68
  :NOP
70:00000000:00000000:00000000:00000000:00000000:000000dd: 0 : 1 : 2 : 3 : 4 : 5 :fffffffe:00000003:68
```

Instruction Implementation:

***Memory:**

lw: load 1, 2, 3, 4, 5, -2, ’hdd to $gr0-gr6 and $ra, can check the value of registers. Here due to the problem of $display, first load should be at pc=4 instead of pc=0 in my display. (First few instructions).

sw: store the value in $gr1-gr7 to registers. (Last few instructions) . A prompt will be displayed when the data are stored to the memory (4 cycles later).

***Arithmetic:**

add: IF: pc = 24; $gr7 = $gr1 + $gr2 = 1 + 2 = 3; Data will be written in $gr7 when pc = 34 (please check: when pc=34 $gr7 = 3); (the followings are the same)

sub: $gr7 = $gr1 - $gr2 = 1 – 2 = -1; (See pc = 38, $gr7 = -1, just below add)

addu: $gr7 = $gr4 + $gr3 = 4 + 3 = 7; (can go through $gr7 to see)

subu: $gr7 = $gr4 - $gr3 = 4 – 3 = 1;

addi: $gr7 = $gr1 - 10 = 1 – ‘h10 = -15 = fffffff1;

addiu: $gr7 = $gr2 + 10 = 2 + ‘h10 = ‘h12;

***Logical**:

and: $gr7 = $gr3 & $gr5 = 0011 & 0101 = 0001 = 1;

or: $gr7 = $gr3 | $gr5 = 0011 | 0101 = 0111 = 7;

nor: $gr7 = $gr3 ~| $gr5 = 0011 ~| 0101 = 1000 (the bits at the front are all 1, negative) = fffffff8;

xor: $gr7 = $gr3 ^ $gr5 = 0011 ^ 0101 = 0110 = 6;

andi: $gr7 = $gr3 & 9 = 0011 & 1001 = 0001 = 1;

ori: $gr7 = $gr3 | 9 = 0011 | 1001 = 1011 = b;

***Shift**:

sll: $gr7 = $gr1 << 1 = 1 << 1 = 2;

srl: $gr7 = $gr2 >> 1 = 2 >> 1 = 1;

sra: $gr7 = $gr6 >>> 1 = -2 >>> 1 = -1 = ffffffff;

sllv: $gr7 = $gr1 << $gr2 = 1 << 2 = 4;

srlv: \$gr7 = \$gr4 >> \$gr1 = 4 >> 1 = 2;

srav: \$gr7 = \$gr6 >>> \$gr1 = -2 >>> 1 = -1 = ffffffff;

slt: \$gr7 = \$gr1 < \$gr2 = 1 < 2 = 1;

***Branch & Jump:**

beq: beq \$gr1, \$gr2, label. 1 $\neq$ 2, so nothing happens.

bne: bne \$gr1, \$gr2, label (offset = 4) ~ pc + 10; 1 $\neq$ 2, so pc will branch to 4*4 = ‘h10 after. You can check that pc=40 → pc=50 after bne executes at pc = 3c;

j: j target - jump to cc. You can check that when pc=9c, j executes, pc=a0 → pc=cc

jr: jr \$ra. You can check that when pc=ec, jr executes. \$ra = 68, pc=f0 → pc=68

jal: jal target - jump to pc=80, save next pc in \$ra. You can check that when pc=64, jal executes, and save pc=68 to \$ra. pc=68 → pc=80

(Jump and branch will make an influence after one clock cycle, in ID, they will send the values back to influence pc, so one more instruction will be executed here)

3 Feeling

When I was writing the project 3, I just felt that project 4 will be hard to finish. However, it seems that it is simpler than what I have thought originally. Maybe we are more familiar with Verilog. Maybe we understand the CPU design better after tutorial. Maybe we have already finished the most important part in project 3 …… Who knows!

Also I didn’t meet many bugs during the project. (Just a problem of the pipeline executing order, cannot use always(*) but always (signals transferred)).

I really gained a lot in these two projects (3.4). It enhanced my understanding of the ALU, control unit, wire connection, and the whole CPU. Also I learned more about how to write the hardware language Verilog code (new language get). The design of the homework is really elaborate. Zan!!!

By the way, TA Micky MA answered many problems in the tutorial, which helped me understand this project a lot. We also had some discussion about the project, course, study, future, life, and so on. During the COVID-19 epidemic period, it’s quite interesting and meaning to have such discussion.

That’s all.