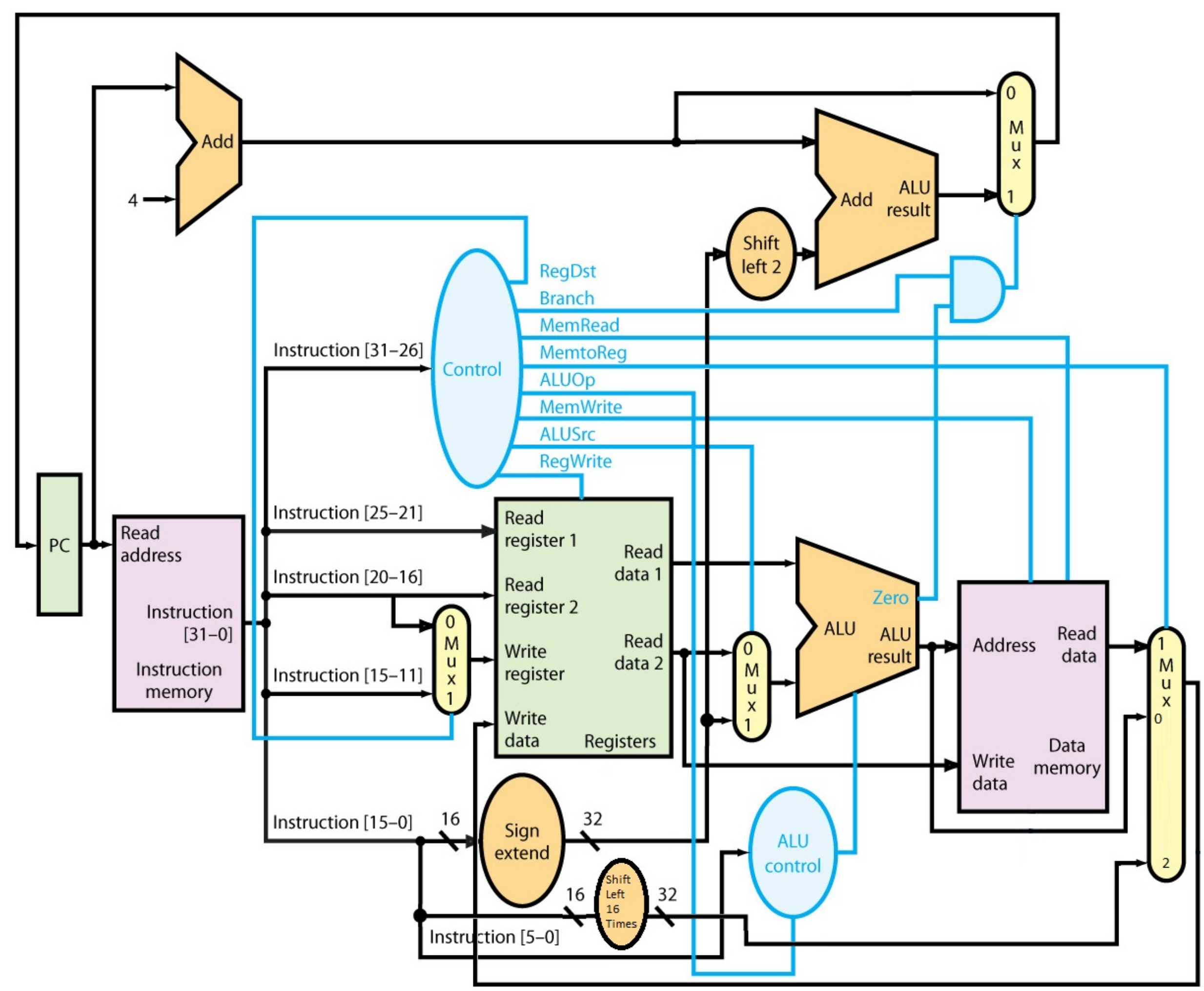
# Proof of Study 6

*Qianlang Chen*

*U1172983*

## Problem 1

Here is the diagram of my modified CPU that supports the *lui* (load upper immediate) instruction:

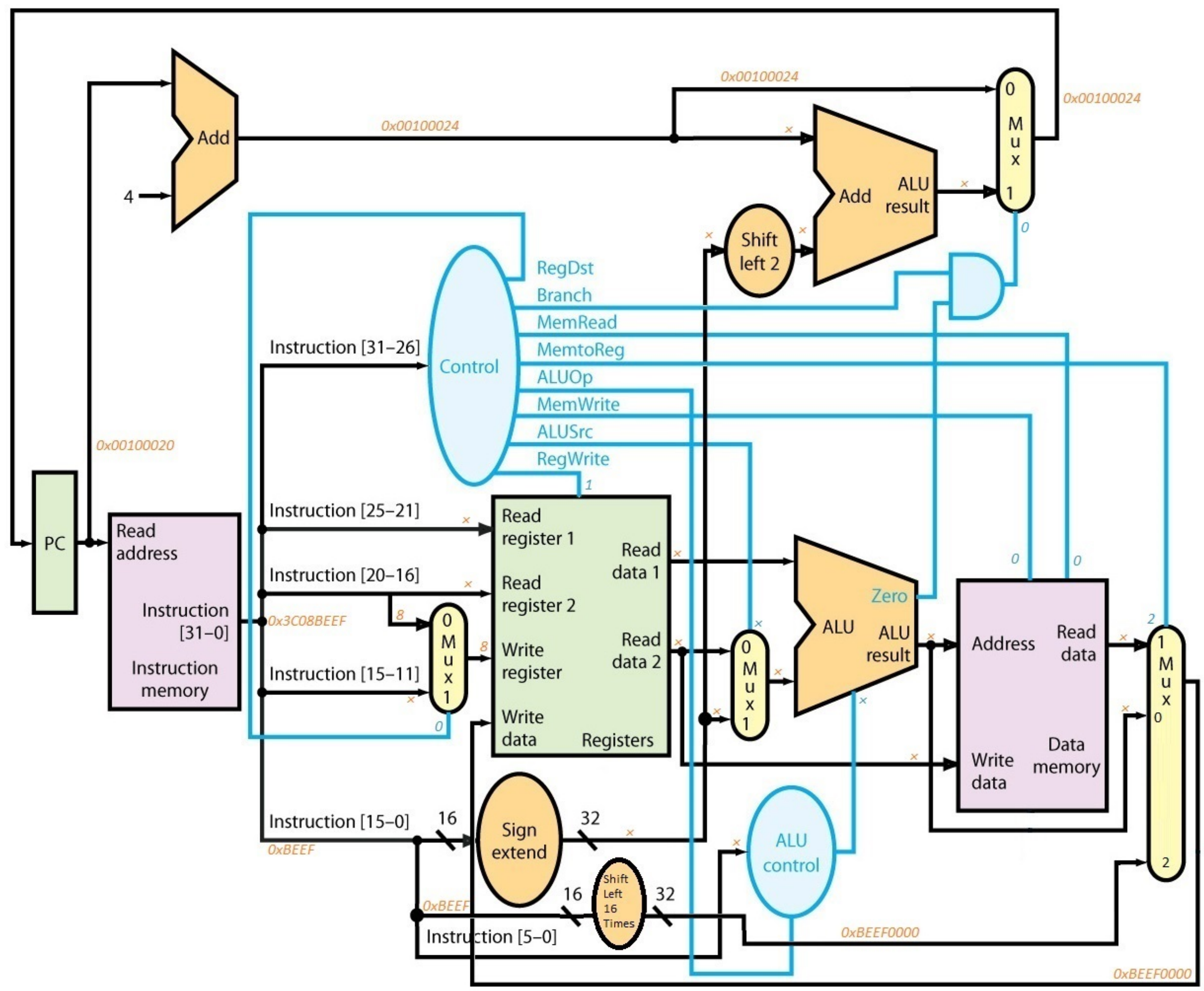


Note that it is just the normal CPU with a little bit of twist on handling the immediate part. It provides two options: extend the sign-bit 16 times (the original option), or shift left 16 times to make the immediate occupy the upper sixteen bits in the result. To perform *lui*, simply send “2” to the rightmost multiplexer *(MemToReg)*, so the resulting value of the second option described above flows through as the write-data. To be exact, these are the control signals needed to run an *lui* instruction:

| RegDst | RegWrite | Branch | ALUSrc | ALUOp | MemRead | MemWrite | MemtoReg |
|---|---|---|---|---|---|---|---|
| × | **1** | **0** | × | × | **0** | **0** | **2** |

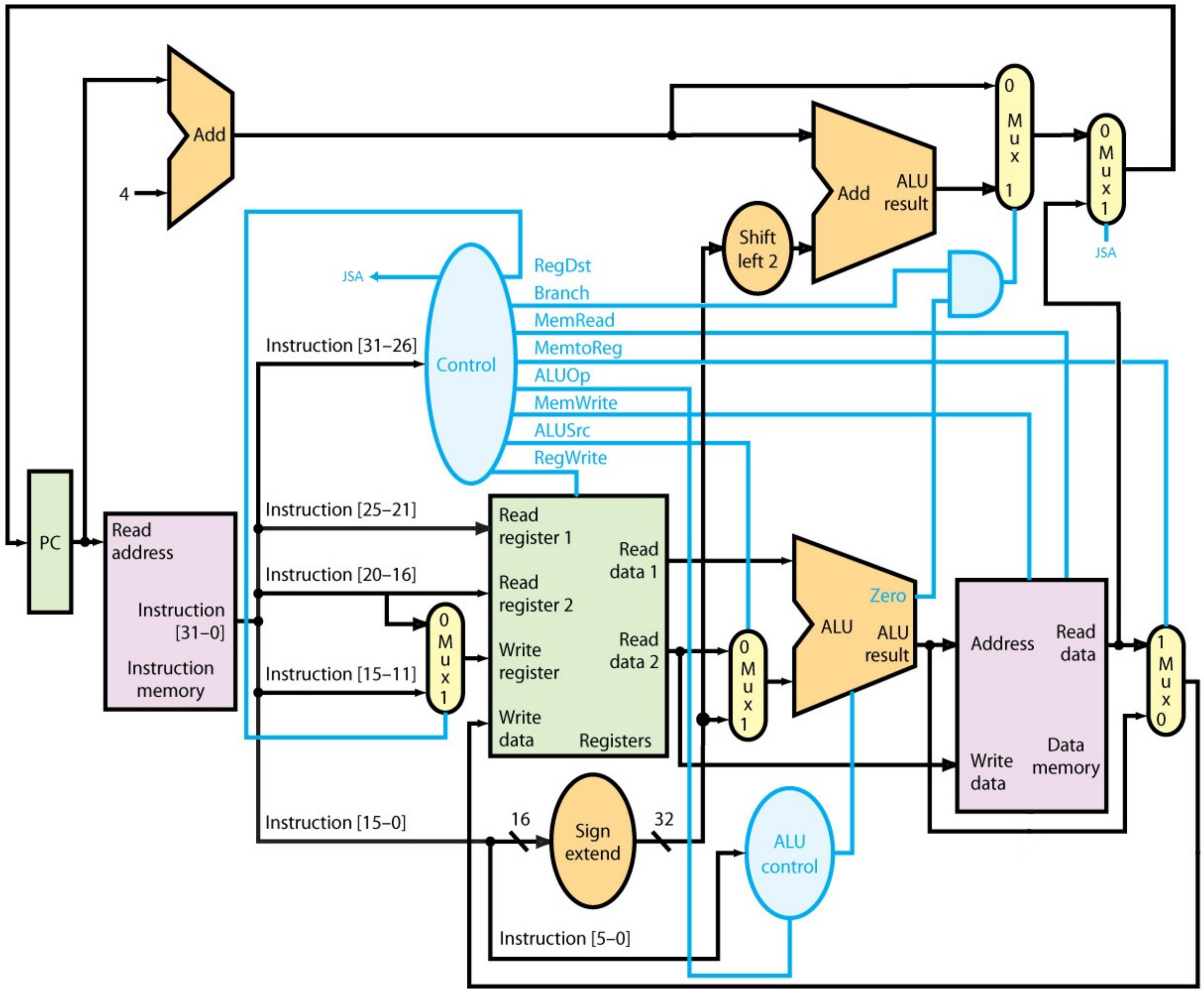
As an example, assuming the program-counter being `0x00100020`, let’s walk through how the following instruction will affect the values in each wire in the CPU: (diagram on the next page, with “×” signs being the *don’t-cares*)

```
lui   $t0,  0xBEEF
```




Finally, to find out the minimum length of time needed for an *lui* instruction to complete, using the assumptions from the study problems, we add up the most costly steps: 200ps for the instruction memory, 30ps for the control box, 30ps for the branch and-gate, 20ps for the *branch* multiplexer, and 10ps for the setup when writing to the program-counter, for a total of 290ps. (Note: the time to run the *lui* instruction is actually shorter than what it takes to advance the program-counter; it takes only 200*(I-mem)*+30*(ctrl)*+20*(memToReg)*+10*(setup)*=260ps to get the result ready.) To give a 10% safety margin, one *lui* instruction takes about 290*(1+10%)=**319 picoseconds** to finish.

## Problem 2

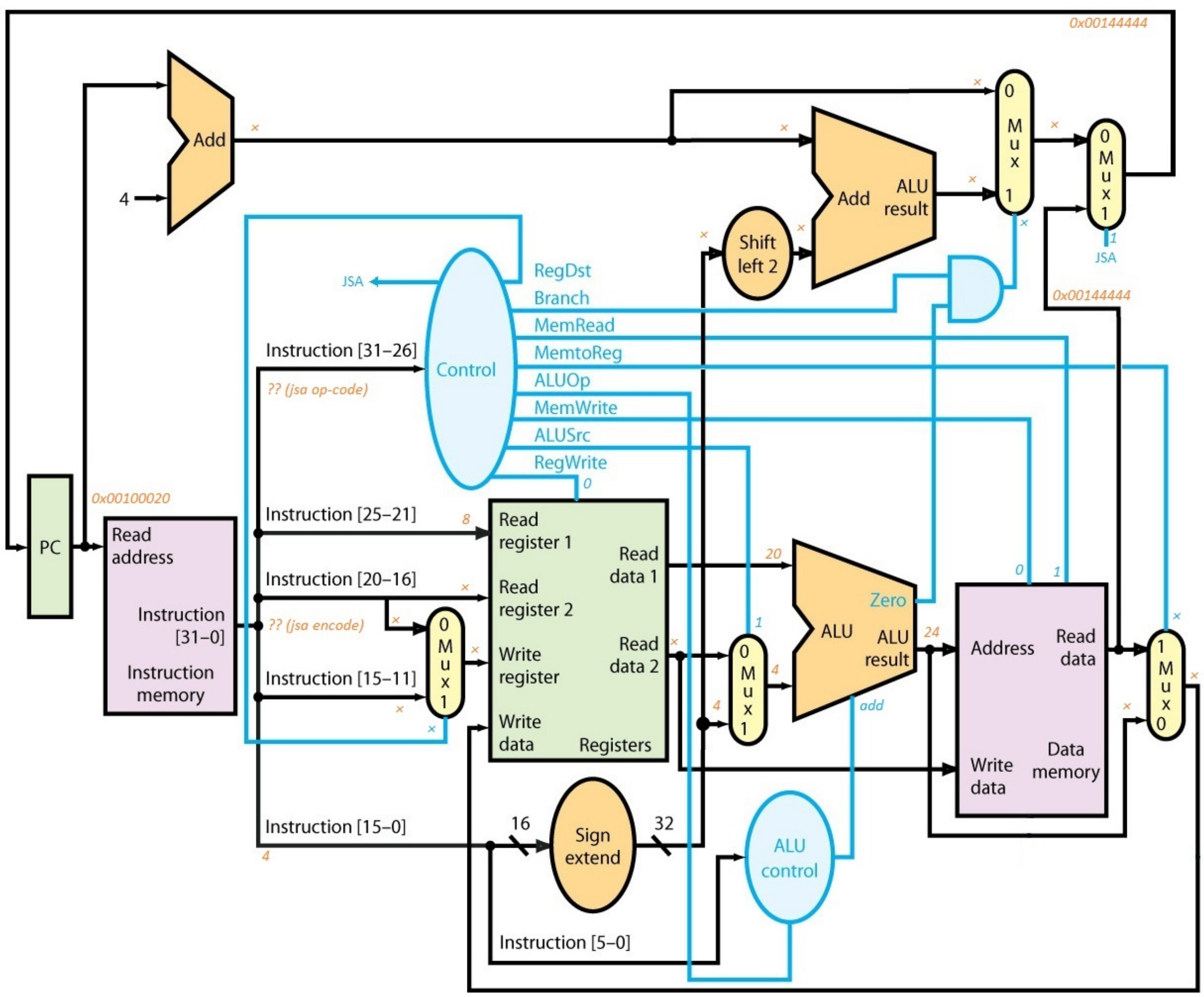



The invented *jsa* (jump stored address) instruction is very similar to the load-word instruction with just a small twist: instead of saving the loaded word into a register, we provide the option (by adding a multiplexer called *JSA* on the top-right) to save the loaded word into the program-counter. Each time the program-counter is advanced, it is to be decided whether to use the advanced counter or a word loaded from the data memory as the new program-counter. Therefore, using this *jsa* instruction is simple and similar to using *lw*:

| RegDst | RegWrite | Branch | ALUSrc | ALUOp | MemRead | MemWrite | MemtoReg | JSA* |
|---|---|---|---|---|---|---|---|---|
| × | 0 | × | 1 | add | 1 | 0 | × | 1 |

To demonstrate what the execution of such instruction looks like, let's use this example (assuming that the program-counter is `0x00100020`, the content of `$t0` is 20, and the content stored in the data memory at location 24 is `0x00144444`):

```
jsa  4($t0)
```

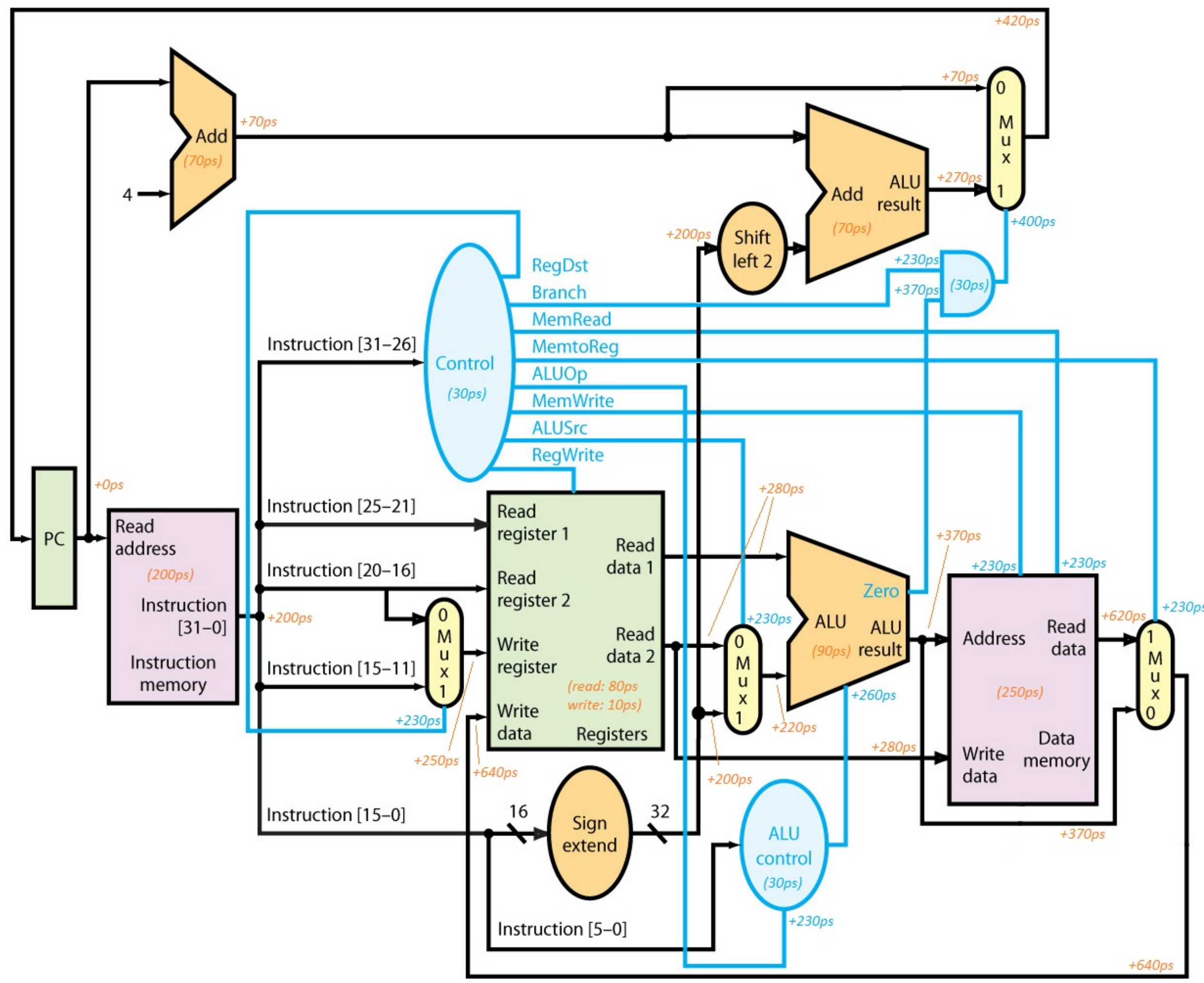



The execution time needed for a *jsa* instruction is 200ps for the instruction memory, 80ps for the register file, 90ps for the ALU, 250ps for the data memory, 20ps for the *JSA* multiplexer, and 10ps for the setup when writing into the program-counter, totaling up to 650 picoseconds. To give a 10% room, one *jsa* instruction finishes in 650*(1+10%)=**715 picoseconds**.

## Problem 3

Since the delay in accessing the data memory is 250 picoseconds and is significantly slower than all other parts in the CPU, in a worst-case scenario, the instruction must access the memory and write to a register (to be as slow as possible). The only instruction that does such a thing is *lw*. Therefore, let's run through a load-word instruction and check its final delay: 200ps for the instruction memory, 80ps for reading the register file, 90ps for the ALU, 250ps for the data memory, 20ps for the *memtoReg* multiplexer, and 10ps for the setup when writing into a register. Note that the delays above are the most costly steps possible at each turning point when running an *lw* instruction, and they add up to 650 picoseconds. For a 10% safety margin, the delay must be as long as 715 picoseconds to guarantee a finish of an *lw* instruction. Therefore, the clock rate of the CPU cannot go faster than **715 picoseconds**, which is equivalent to about **1.40 GHz**.

The diagram on the next page shows the delay for each wire in the worst-case. Note that each step must wait for the slowest input to arrive before execution. The delay for the *Write Data* input of the register file is 640ps, so adding a 10ps delay to the setup time gives a total delay of 650ps.



PC
+0ps
Add
(70ps)
+70ps
4
+420ps
+70ps
0
Mux
1
+270ps
+400ps
Add
ALU
result
(70ps)
Shift
left 2
+200ps
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
(30ps)
+230ps
+370ps
(30ps)
Instruction [31–26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read
address
(200ps)
Instruction
[31–0]
Instruction
memory
+200ps
0
Mux
1
+230ps
+250ps
+640ps
Read
register 1
Read
register 2
Write
register
Write
data
Read
data 1
Read
data 2
(read: 80ps
write: 10ps)
Registers
+280ps
+230ps
0
Mux
1
+220ps
+200ps
Zero
ALU
ALU
result
(90ps)
+260ps
+370ps
+230ps
+230ps
Address
Read
data
(250ps)
Write
data
Data
memory
+280ps
+370ps
+620ps
+230ps
1
Mux
0
16
Sign
extend
32
ALU
control
(30ps)
+230ps
+640ps