Computer Architecture Final Project Report

B09901021 鄭定緯 B09901191 魏振宇

1. Record of execution cycle number of each instruction set:

Instruction	Execution cycle
I0	98
I1	649
I2	226
I3	442

2. Register Table:

Register Name	Type	Width	Bus	MB	AR	AS	SR	SS	ST
PC_reg	Flip-flop	31	Y	N	Y	N	N	N	N
PC_reg	Flip-flop	1	N	N	N	Y	N	N	N
state_reg	Flip-flop	4	Y	N	Y	N	N	N	N

Register Name	Type	Width	Bus	MB	AR	AS	SR	SS	ST
mem_reg	Flip-flop	995	Y	N	Y	N	N	N	N
mem_reg	Flip-flop	29	Y	N	N	Y	N	N	N

Register Name	Type	Width	Bus	MB	AR	AS	SR	SS	ST
operand_reg_reg	Flip-flop	32	Y	N	N	N	N	N	N
counter_reg	Flip-flop	6	Y	N	N	N	N	N	N
state_reg	Flip-flop	4	Y	N	Y	N	N	N	N
shift_reg_reg	Flip-flop	64	Y	N	Y	N	N	N	N

Register Name	Type	Width	Bus	MB	AR	AS	SR	SS	ST
state_reg	Flip-flop	2	Y	N	Y	N	N	N	N

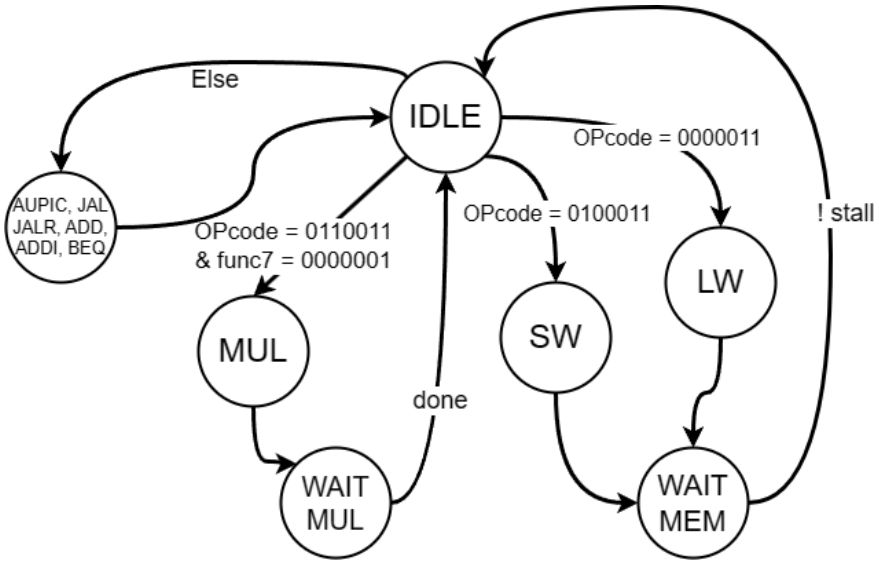
Register Name	Type	Width	Bus	MB	AR	AS	SR	SS	ST
read_hit_reg	Flip-flop	1	N	N	Y	N	N	N	N

Register Name	Type	Width	Bus	MB	AR	AS	SR	SS	ST
read_hit_data_reg	Flip-flop	32	Y	N	Y	N	N	N	N

We can see that all the register type are Flip-flop.

3. CPU architecture:

The following figure is our FSM of CPU:



According to the finite state machine depicted above, our ALU would follow the current state to perform calculations, while the CPU would wait if it requires access to memory or if it need to wait multi cycles.

The structure of the other components is the same as the figure shown in the lecture slides.

4. Describe how you design the data path of instructions not referred in the lecture slides

JAL: We use an adder to calculate pc + imm, and we make the mux choose pc + imm instead of pc + 4 , and then we update pc .

JALR: We read rdata1 from register file, then we use an adder to calculate rdata1+ imm, and we make the mux choose rdata1+ imm instead of pc + 4, and then we update pc .

AUIPC: We input pc and imm to ALU and let it calculate pc + {imm,12'b0}, then we write back to rd with the value of ALU output.

5. Describe how you handle multi-cycle instructions

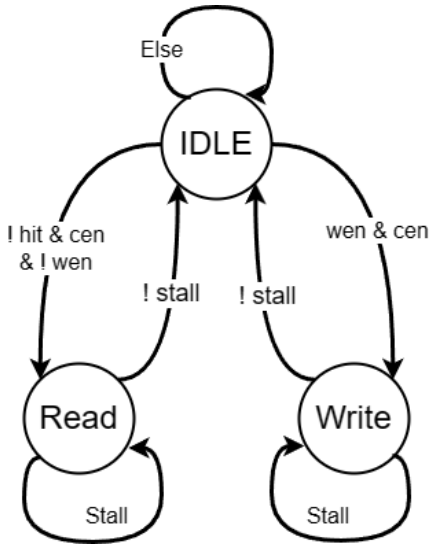
As described above, when the CPU needs to wait for the execution of a multiplication instruction, the current state will transition to WAIT_MUL and wait for the multiplier. Once the multiplier completes its operation, it will raise the 'done' signal, and in the next cycle, the state will transition to IDLE.

When CPU need to wait for memory, the current state will transition to WAIT_MEM, and once the memory is ready, it will set the 'stall' signal to zero, and in the next cycle, the state will transition to IDLE.

6. Cache Design

Our cache is using direct mapped with 128 blocks, 1 word, and using write through policy, in every block, there have 56 bits(1 Valid bit, 23 Tag bits and 32 Data bits).

The following figure is our FSM of cache:



The table below demonstrates how our cache improves time performance :

Instruction	Execution cycle before cache	Execution cycle after cache
I0	98	98
I1	809	649
I2	296	226
I3	662	442

In I3 instruction, we can see that the speed up is $662/442 = 1.50$.

7. Observation

When we design cache, we found that if we adjust the block size of cache to 256 or larger, the performance would be same as that of the block size of 128.

8. Work distribution table

鄭定緯	CPU, report.
魏振宇	Cache, report.