EECS151 ASIC Project Report

Timothy Chu
Yu(Jaray) Li

May 8, 2024

Problem 1: Team Information

Please list the full name of the individuals. If you worked only, then write your name down as *Project Partner 1* and leave *Project Partner 2* blank.

Project Partner 1: Timothy Chu

Project Partner 2: Yu(Jaray) Li

Problem 2: Design

2.1 Summary:

Core Functional? (Yes/No): Yes

Number of Pipeline Stages: 3 stages

Forwarding Implemented? (Yes/No): Yes

Branch Prediction Implemented? (Yes/No): Yes

Cache Implemented? (Yes/No): Yes

Cache Functional? (Yes/No): No

Cache Size: N/A

Cache Associativity: N/A

Cache Read Miss Latency: N/A

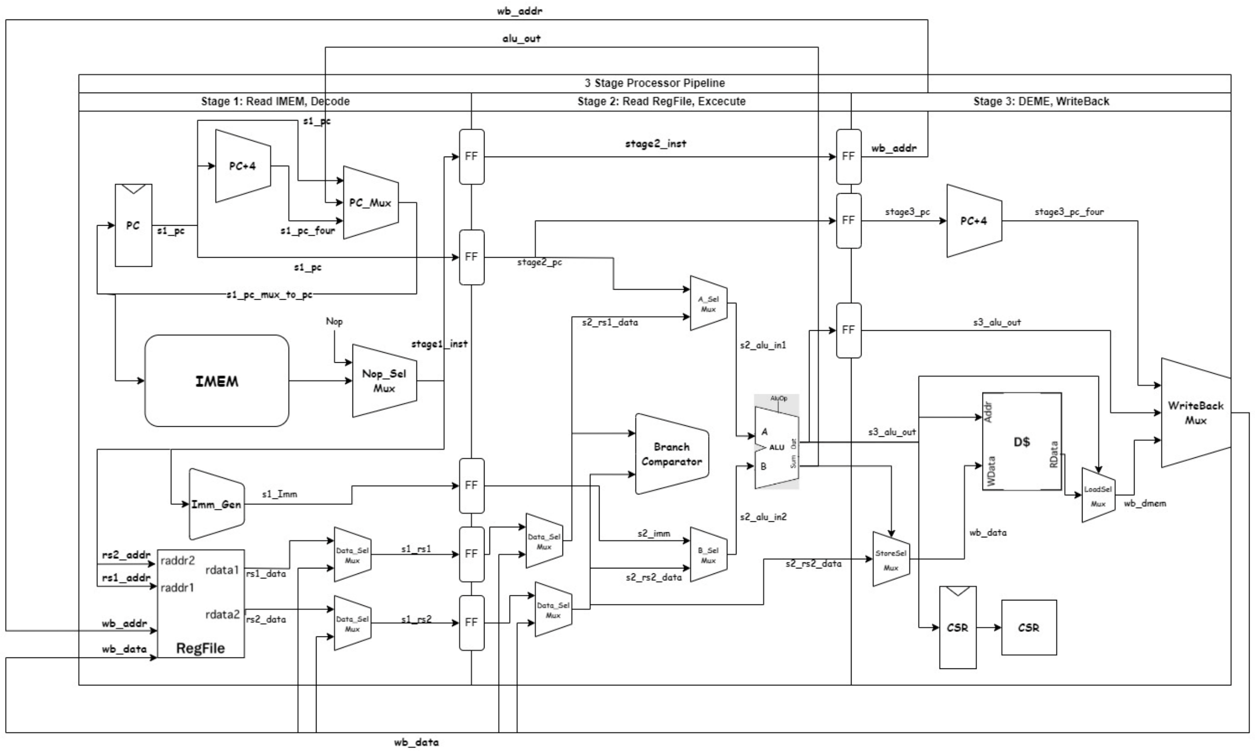
Cache Read Hit Latency: N/A

Cache Write Miss Latency: N/A

Cache Write Hit Latency: N/A

Other Optimization (Yes/No; please describe in later section): No

2.2 Pipeline Design



We divide the pipeline architecture into three distinct stages: Instruction Fetch and Decode (IF/ID), Register Read and Execute (ID/EX), and Data Memory Access and Write Back (EX/WB).

In the initial stage, IF/ID, the processor fetches instructions from the Instruction Memory (IMEM) and decodes them to generate control signals necessary for subsequent execution. Simultaneously, the program counter (PC) is incremented to point to the next instruction, while the PC+4 is computed for branch calculations. Furthermore, the Register File is accessed to retrieve operand values, and immediate values are generated for instructions requiring them. This stage also handles the insertion of No Operation (NOP) instructions when necessary to maintain the pipeline's rhythm.

Moving to the second stage, ID/EX, the processor reads operand values from the Register File and performs the execution of ALU operations and branch comparisons. The branch comparator facilitates conditional branching by comparing values as required by branch instructions. Throughout this stage, hazards, particularly data hazards, are managed through mechanisms like forwarding, enabling the smooth flow of data through the pipeline. Forwarding allows data to be passed from one stage to another without waiting for it to be written back to the register file, thus minimizing stalls and enhancing pipeline efficiency.

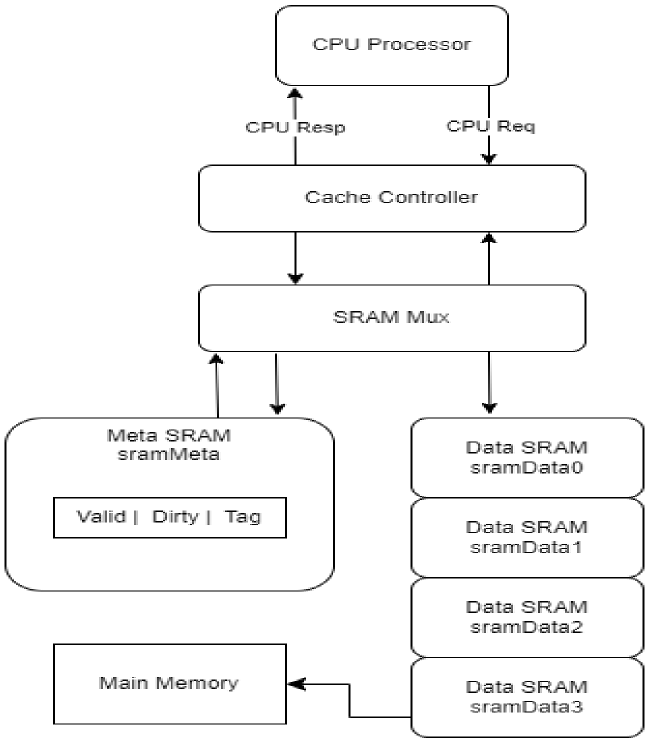
Lastly, the third stage, EX/WB, involves accessing the Data Memory (DMEM) for load and store operations, selecting data to be written back to the register file using a writeback mux, and handling Control and Status Register (CSR) operations. Here, any remaining data hazards are addressed, ensuring that the pipeline operates smoothly. Overall, the pipeline's architecture is designed to maximize performance by efficiently managing instruction execution and handling hazards that may arise during the process.

2.3 Cache Design

[Please insert diagram of your cache structure here. Please include a state transition diagram of your controller.]

Explain your cache at high level. Be sure to state which SRAM macros you used, what metadata is used, and how your cache controller works.

If you have implemented any optimizations beyond those already mentioned, then please discuss them below:



The cache implemented in this design follows a direct-mapped architecture with 64 lines (LINES=64) and a line size of 512 bits (CACHE_LINE_SIZE). The cache is designed to efficiently handle CPU requests by minimizing access latency and optimizing performance.

Cache Structure:

The cache consists of separate SRAM arrays for storing metadata and data. The metadata SRAM (sramMeta) is a dual-port SRAM that holds the valid bit, dirty bits, and tag bits for each cache line. The data is stored across four dual-port SRAM arrays (sramData0-sramData3), enabling parallel access and reducing access latency.

The separation of metadata and data allows for efficient management of cache lines and simplifies the cache controller design. The metadata SRAM is accessed using the index bits of the CPU request address, while the data SRAMs are accessed using a combination of the index and the word select bits.

Cache Controller:

The cache controller is responsible for handling CPU requests and managing the cache state. It operates in five distinct states: IDLE, READ_QUERY, WRITE_QUERY, CACHE_READ_MISS, and CACHE_WRITE_MISS. The state transitions are determined by the current state and the CPU request type (read or write).

When the CPU issues a read request, the cache controller enters the READ_QUERY state. If the requested data is found in the cache (a hit), the data is directly returned to the CPU. If the data is not found (a miss), the controller transitions to the CACHE_READ_MISS state, initiating a memory request to fetch the missing data.

Similarly, for a write request, the cache controller enters the WRITE_QUERY state. If the write hits in the cache, the data is updated in the corresponding cache line, and the dirty bit is set. If the write misses, the controller transitions to the CACHE_WRITE_MISS state. In case of a write miss with a dirty victim cache line, the controller first writes back the dirty line to memory before proceeding with the miss handling.

Performance Optimizations:

Data Bypassing: In case of a read miss, the cache controller immediately saves the requested data from memory once it arrives, allowing the CPU request to be served without waiting for the entire cache line to be filled.

Victim Cache Line Handling: When a write miss occurs and the victim cache line is dirty, the controller efficiently manages the write-back process to minimize stalls in the pipeline.

Parallel Data Filling: During a cache miss, the controller fills the four data SRAMs in parallel, reducing the overall miss penalty.

These optimizations aim to reduce the average access latency and improve the overall system performance.

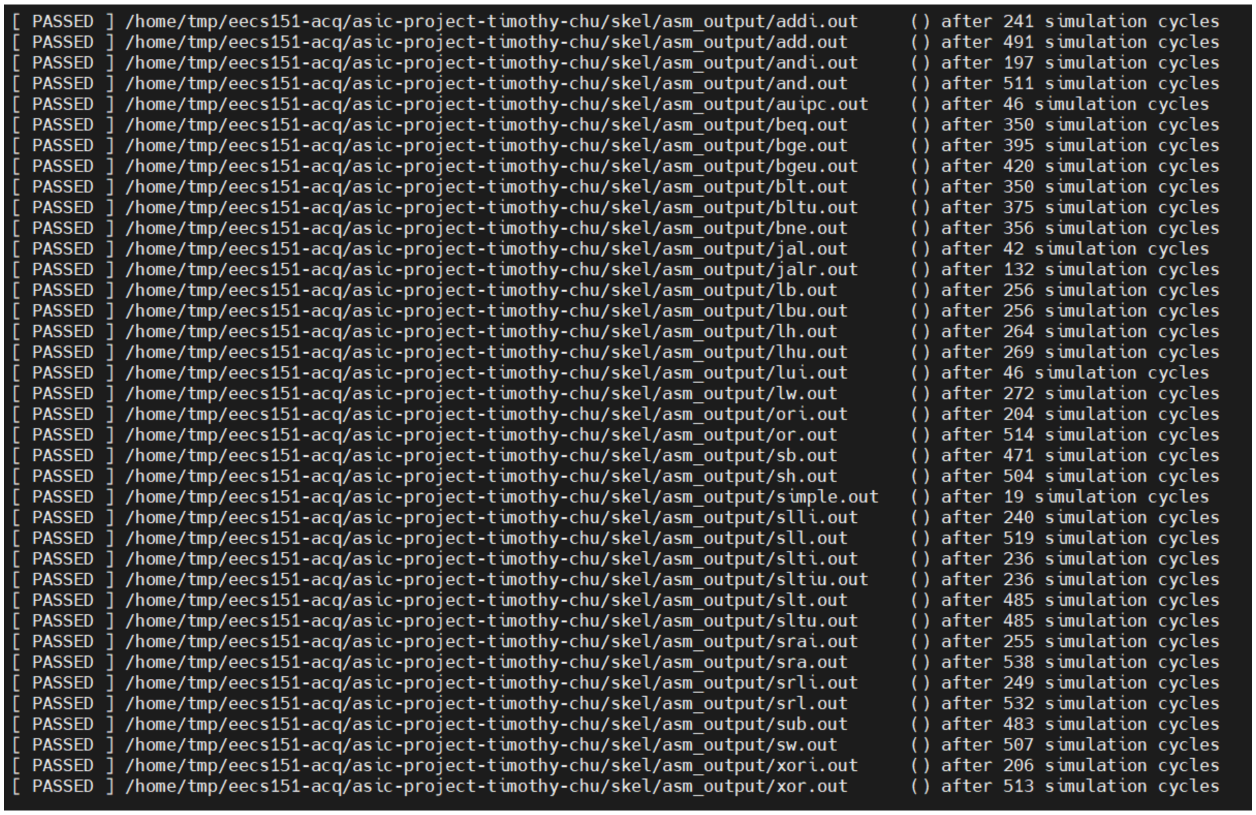
Performance Analysis:

The cache design achieves a balance between performance and cost. The hit latency for read and write operations is relatively low, as the data can be directly accessed from the cache. The miss penalty, however, is higher due to the need to fetch data from memory and potentially write back dirty lines.

Problem 3: Simulation

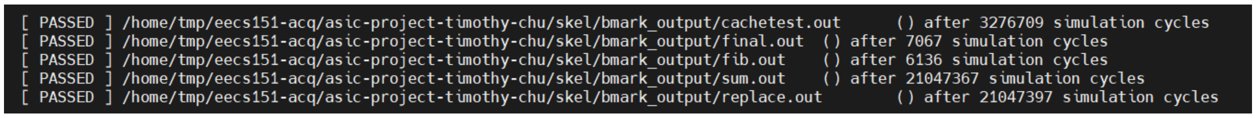
3.1 Behavioral

Did you pass all the assembly tests (*asm* tests)? If not, please list the tests you failed and why (if known). Include a screenshot of the passing ASM tests.



```
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/addi.out     () after 241 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/add.out      () after 491 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/andi.out     () after 197 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/and.out      () after 511 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/auipc.out    () after 46 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/beq.out      () after 350 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/bge.out      () after 395 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/bgeu.out     () after 420 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/blt.out      () after 350 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/bltu.out     () after 375 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/bne.out      () after 356 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/jal.out      () after 42 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/jalr.out     () after 132 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/lb.out       () after 256 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/lbu.out      () after 256 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/lh.out       () after 264 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/lhu.out      () after 269 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/lui.out      () after 46 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/lw.out       () after 272 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/ori.out      () after 204 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/or.out       () after 514 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/sb.out       () after 471 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/sh.out       () after 504 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/simple.out   () after 19 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/slli.out     () after 240 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/sll.out      () after 519 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/slti.out     () after 236 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/sltiu.out    () after 236 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/slt.out      () after 485 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/sltu.out     () after 485 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/srai.out     () after 255 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/sra.out      () after 538 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/srli.out     () after 249 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/srl.out      () after 532 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/sub.out      () after 483 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/sw.out       () after 507 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/xori.out     () after 206 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/asm_output/xor.out      () after 513 simulation cycles
```

Did you pass all the benchmark tests (*bmark* tests)? If not, please list the tests you failed and why (if known). Include a screenshot of the passing bmark tests.



```
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/bmark_output/cachetest.out      () after 3276709 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/bmark_output/final.out  () after 7067 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/bmark_output/fib.out    () after 6136 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/bmark_output/sum.out    () after 21047367 simulation cycles
[ PASSED ] /home/tmp/eecs151-acq/asic-project-timothy-chu/skel/bmark_output/replace.out        () after 21047397 simulation cycles
```

List the reported cycles for each test (both assembly and benchmark). Leave the corresponding cell blank if you failed a test.

addi	241	*lb*	256	*slti*	236
add	491	*lbu*	256	*sltiu*	236
andi	197	*lh*	264	*slt*	485
and	511	*lhu*	269	*sltu*	485
auipc	46	*lui*	46	*srai*	255
beq	350	*lw*	272	*sra*	538
bge	395	*ori*	204	*srli*	249
bgeu	420	*or*	514	*srl*	532
blt	350	*sb*	471	*sub*	483
bltu	375	*sh*	504	*sw*	507
bne	356	*simple*	19	*xori*	206
jal	42	*slli*	240	*xor*	513
jalr	132	*sll*	519		

Table 1: Reported cycles for assembly tests

cachetest	3276709
final	7067
fib	6136
sum	21047367
replace	21047397

Table 2: Reported cycles for benchmark tests

Problem 4: Implementation: Synthesis

4.1 Timing

Does your design pass timing? (Yes/No):

Maximum frequency for which the design passes timing (MHz):

Slack on Critical Path (ps):

Critical Path Location (can list start and end point):

Explain why the critical path is located where it is given your implementation. If the start and end points of your critical path do not inform the function in your design, then please provide context. This answer should be implementation specific (ex. our forwarding path the ALU, removing a register reduce miss latency).

Problem 5: Implementation: Place-and-Route

5.1 Floorplan

[*Please insert a screenshot of your floorplan here*] Explain your

floorplanning below.

5.2 Timing

Does your design pass timing? (Yes/No):

Maximum frequency for which the design passes timing (MHz):

Slack on Critical Path (ps):

Critical Path Location (can list start and end point):

[*Please insert a screenshot of your clock tree*]

Discuss the quality of your clock tree below.

[*Please insert screenshots of your setup and hold slack critical paths from the Innovus timing reports*]

Explain why this is the critical path given your implementation. Make sure to state whether it is the same as the critical path from synthesis.

5.3 Area

[*Please insert a screenshot of the output from the report _area command in Innovus*]

What module consumes the most area (the area should be seen in the screenshot above). Explain why below

Problem 6: Known Bugs

If you have any known bugs in the core (not including the cache) list them and what you believe is the root cause.

N/A

If you have any known bugs in the cache list them and what you believe is the root cause.

N/A

Problem 7: Optimizations (Optional)

Explain any optimizations you made for either performance or area.

If you would like to be included in the design competition, include your performance-area score for the *sum* benchmark:

$$Score = ClockPeriod * Cycles * Area^{\frac{1}{2}}$$

The units for *ClockPeriod* and *Area* should be *ns* and *µm*. Please refer to Checkpoint 4 for more specifics and an example. List the values for each variable.

ClockPeriod (ns)	
Cycles	
Area (µm)	

Score for *sum* benchmark:

Problem 8: Extra Information

Your final grade will be a reflection of your design complexity, functionality, and any other information the TAs have about your submission. If there is anything the TAs should know about while grading your design (not team dynamics or please discuss them below.

We passed the asm and bmark tests but encountered problems during synthesis and PAR. We first used the processor with cache for synthesis but it failed. But when we tried to adjust the yml file and use the processor without cache it still failed to pass the test.

I think there was a problem when adjusting the yml file, and it did not cover all the required content. Another problem is that the integration of the original code and the cache added later do not fully correspond, which allows us to pass rtl but cannot pass syn and par.

Problem 9: Feedback

If you have any feedback on how to improve the project please tell us below. For example, do you feel the labs prepared you to complete the project.

I think labs are very helpful in completing the project. Through practical operations and exercises, we are able to gain a deeper understanding and application of the concepts and skills learned. In addition, many office hours and TA's help and explanations made the entire project less difficult. Therefore, I think the laboratory is one of the key factors for our successful completion of the project.