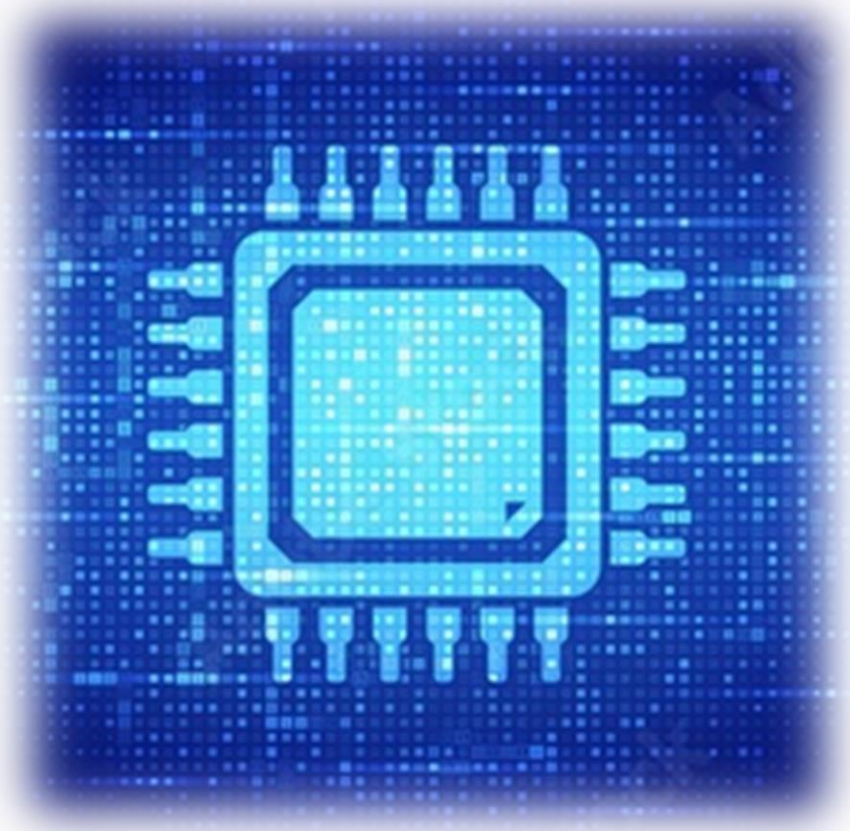
CISC Simulator in C++

Butnaru Alexandru



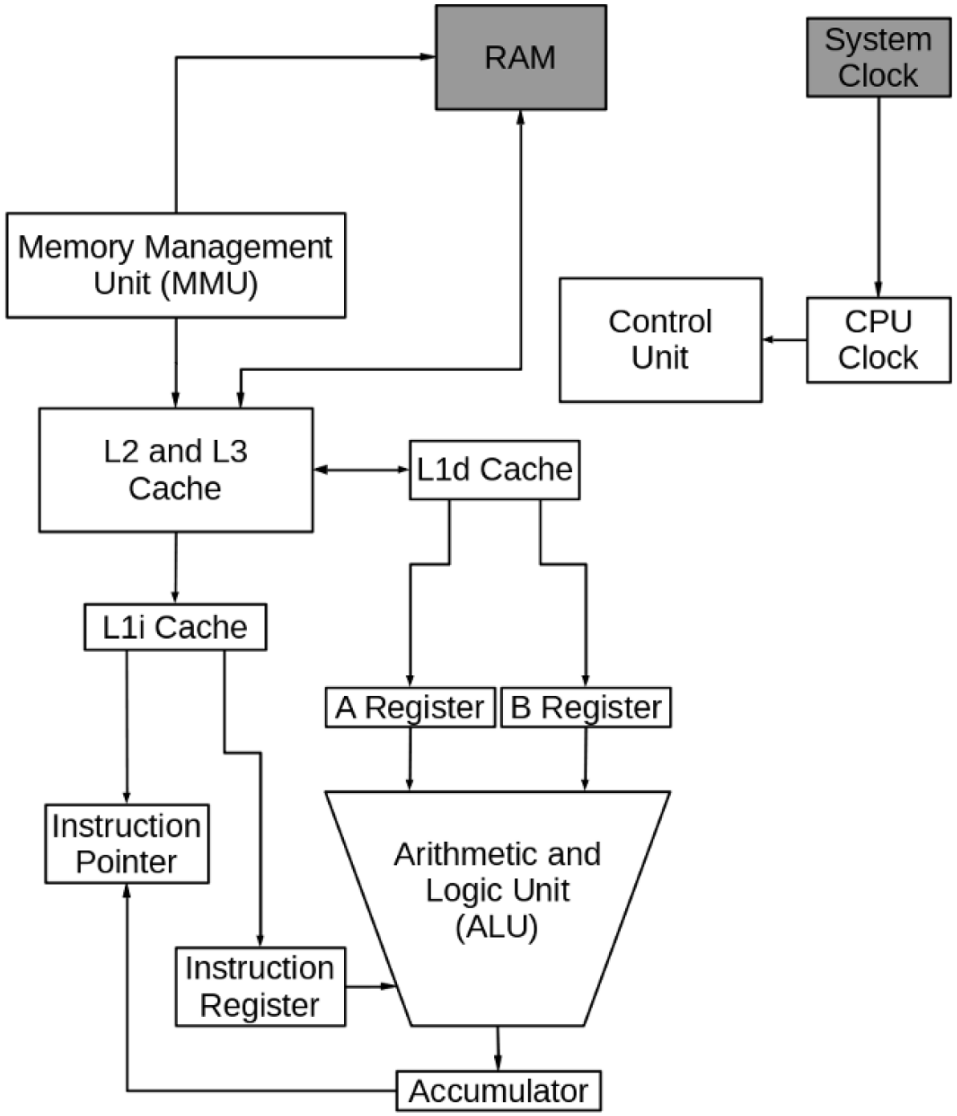
Mentors:
- Sandulean Vlad
- Uleru George-Iulian
- Coman Sergiu
- Rotaru Cristian

2024 @ AMD

together we advance_

Glossary

Motivation



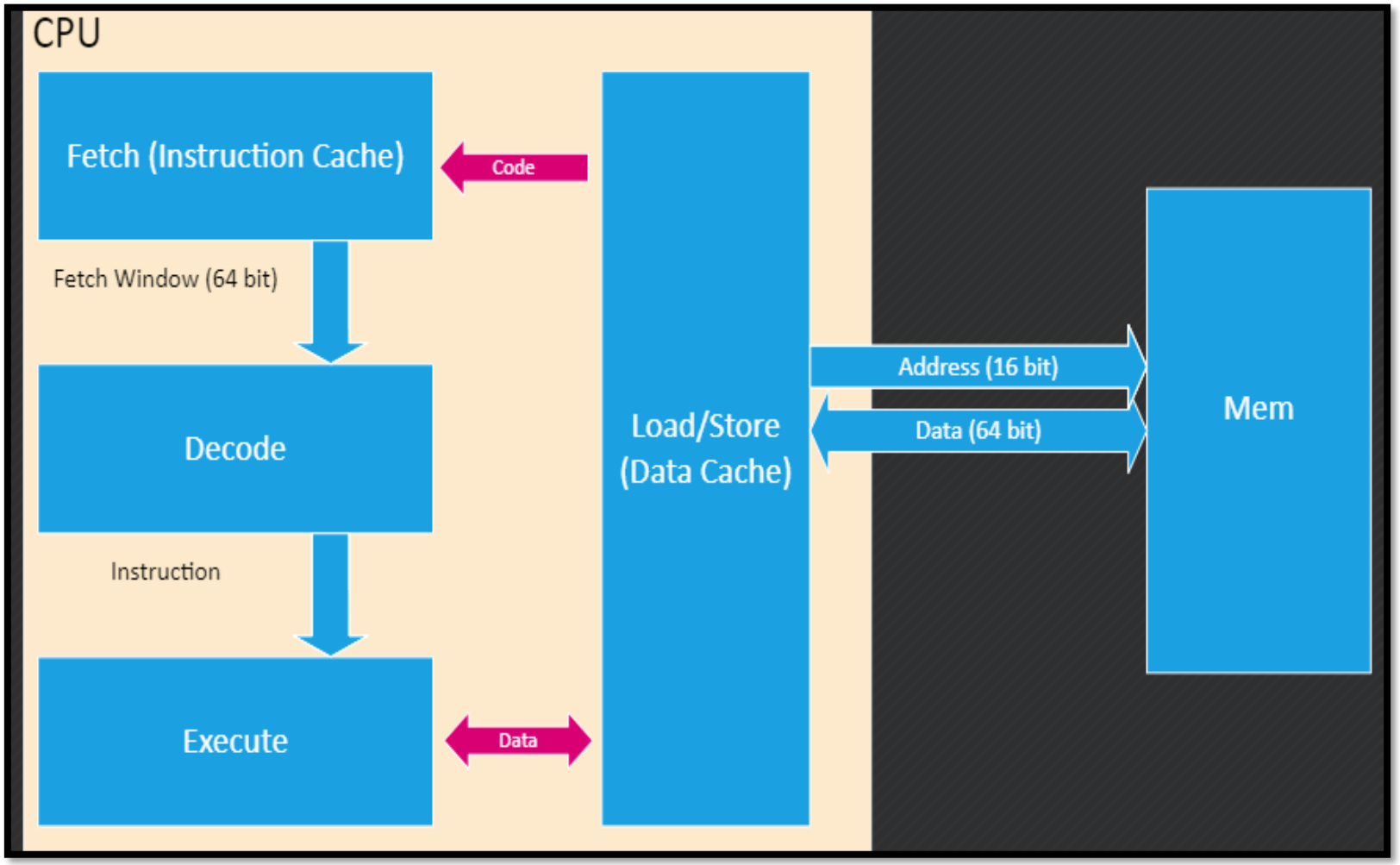
* Image not fully representative of implemented architecture

Knowledge of low-level modules & programming is essential for the understanding of high-level Computer Science.

Driven by curiosity and the desire to broaden my computer science skills into this field, too.

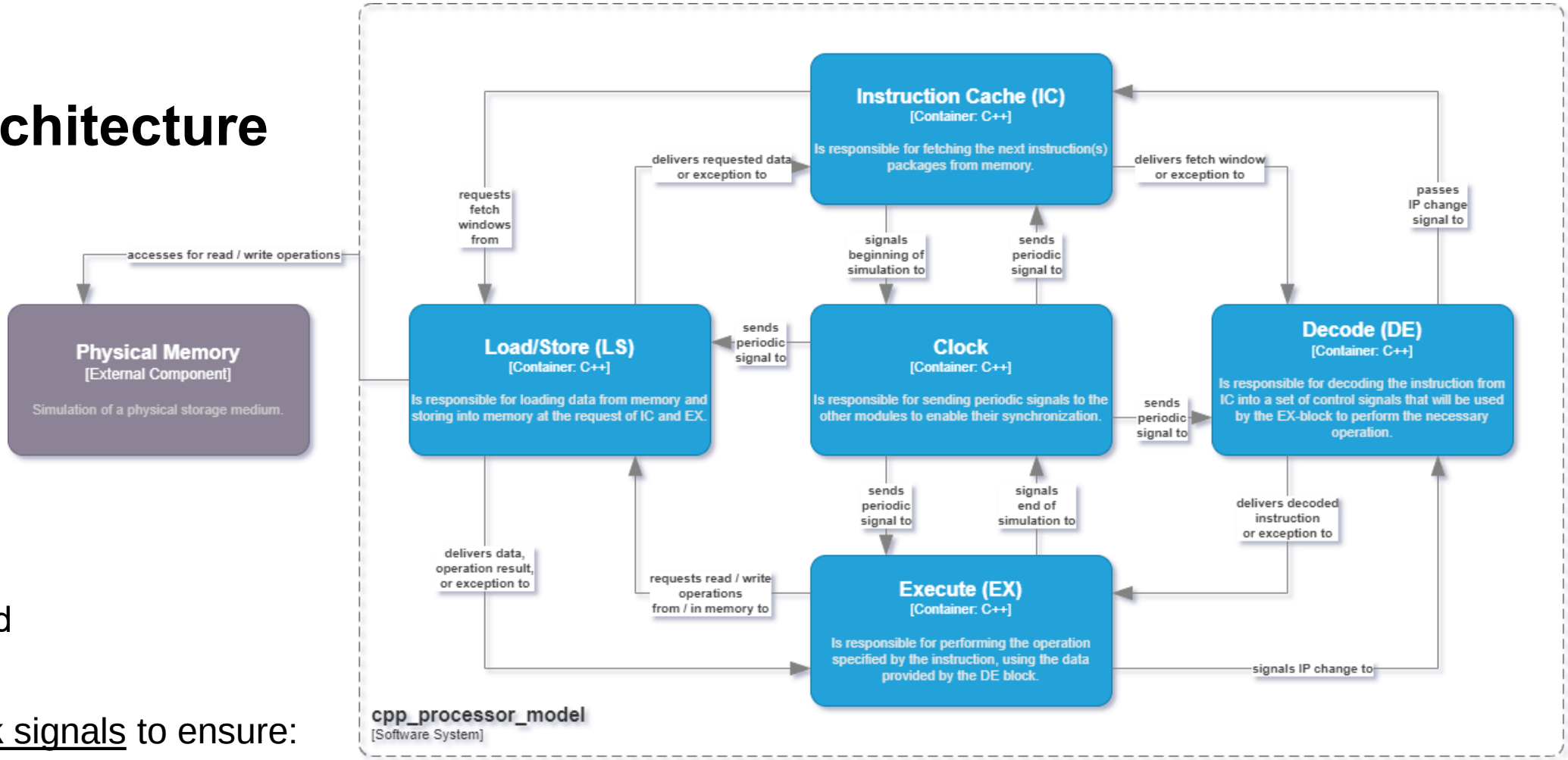
The end goal of this project is providing an observational tool for the execution of assembly-level instructions.

General Architecture, Parameters, Restrictions



- 4 parallel modules
- Each module has a set number of active cycles
- Modules are synchronized by a clock
- **16-bit** address width
- **16-bit** data size – "*words*"
- **64-bit** raw hex code deliveries from memory – "*fetch windows*"
- Deterministic – no variation allowed in the execution times between runs
- Physical memory – simulated, exterior to the CPU

Project's Architecture



- 1 module = 1 thread

- All relying on **Clock** signals to ensure:

1) running a set amount of time per operation cycle
2) no data race occurring between modules

The Simulated Assembly Language

```
count_1_bits_loop:
    cmp [param2], 0
    jz count_1_bits_end
    div [param2], 2
    cmp r1, 0
    jg count_1_bits_increment
    mov [param2], r0
    jmp count_1_bits_loop

count_1_bits_increment:
    add r2, 1
    mov [param2], r0
    jmp count_1_bits_loop

count_1_bits_end:
    mov [result], r2
    ret
```

Instruction set:

- Arithmetic operators: **add**, **sub**, **mul**, **div**
- Comparison operator **cmp -** updates flags
- Execution flow control:
  - labels
  - **jmp**
  - conditional jumps **je**, **jg**, **jl**, **jz** based on the current `equal`, `greater`, `zero` flags' state
  - function **call** & **ret**
- **mov** for assignment & storage of data
- Variable declaration in .data zone: **dw**, **dblock**
- Stack **push** & **pop**

Provided assembler program for turning assembly code into hex input file.

Running the program:

./cpp_processor_model <inputSourceCode.hex> [outputFile | null for console output] [memoryDumpFile if outputFile mentioned]

Execution Example

(snippet)

Assembly Code

```
push [param_n]
sub [param_n], 1
call fib
push [glob_var_a]
```

assembler

Hex Input

```
4040
1054
0841
1054
0001
3420
100a
```

CPU simulator

```
[DE@T=186]> Decoded 'sub [1054],  1' from #102a
[LS@T=187]> Stored { 0006 } at #ffd8 for EX
    (Entirely using LS's cache)
[LS@T=188]> Accepted request from IC regarding #1030
[EX@T=190]> Finished executing: push [1054] (push 6 )
[EX@T=191]> Began executing 'sub [1054],  1' from #101c
[LS@T=202]> Fetched [3420 100a 4440 1058] from #1030 for IC
[LS@T=203]> Accepted request from EX regarding #1054
[IC@T=204]> Delivered [3420 100a 4440 1058] from #1030 to DE
[IC@T=205]> Requested fetch window starting at #1038 from LS
[DE@T=206]> Decoded 'call 4106 ' from #1030
[DE@T=208]> Decoded 'pop [1058] ' from #1034
[LS@T=209]> Fetched { 0006 } from #1054 for EX
    (Entirely using LS's cache)
[LS@T=210]> Accepted request from IC regarding #1038
[LS@T=224]> Fetched [0442 1056 1058 4440] from #1038 for IC
[LS@T=225]> Accepted request from EX regarding #1054
[IC@T=226]> Delivered [0442 1056 1058 4440] from #1038 to DE
[IC@T=227]> Requested fetch window starting at #1040 from LS
[DE@T=228]> Decoded 'add [1056],  [1058]' from #1038
[LS@T=231]> Stored { 0005 } at #1054 for EX
    (Entirely using LS's cache)
[LS@T=232]> Accepted request from IC regarding #1040
[EX@T=234]> Finished executing: sub [1054],  1 ([1054] = 5)
[EX@T=235]> Began executing 'call 4106 ' from #1022
[LS@T=246]> Fetched [1054 3800 0c41 1056] from #1040 for IC
[LS@T=247]> Accepted request from EX regarding #ffc4
[IC@T=248]> Delivered [1054 3800 0c41 1056] from #1040 to DE
[IC@T=249]> Requested fetch window starting at #1048 from LS
[DE@T=250]> Decoded 'pop [1054] ' from #103e
[DE@T=252]> Decoded 'ret  ' from #1042
[LS@T=253]> Stored { 0000 0000 0000 0000 0000 0000 0000 0000 2000 1026 } at #ffc4 for EX
    (Entirely using LS's cache)
[LS@T=254]> Accepted request from IC regarding #1048
[EX@T=257]> Finished executing: call 4106
Saved state:
    IP = #1026
    Flags.Z=0 Flags.E=0 Flags.G=1
    Registers: r0=0 r1=0 r2=0 r3=0 r4=0 r5=0 r6=0 r7=0
```

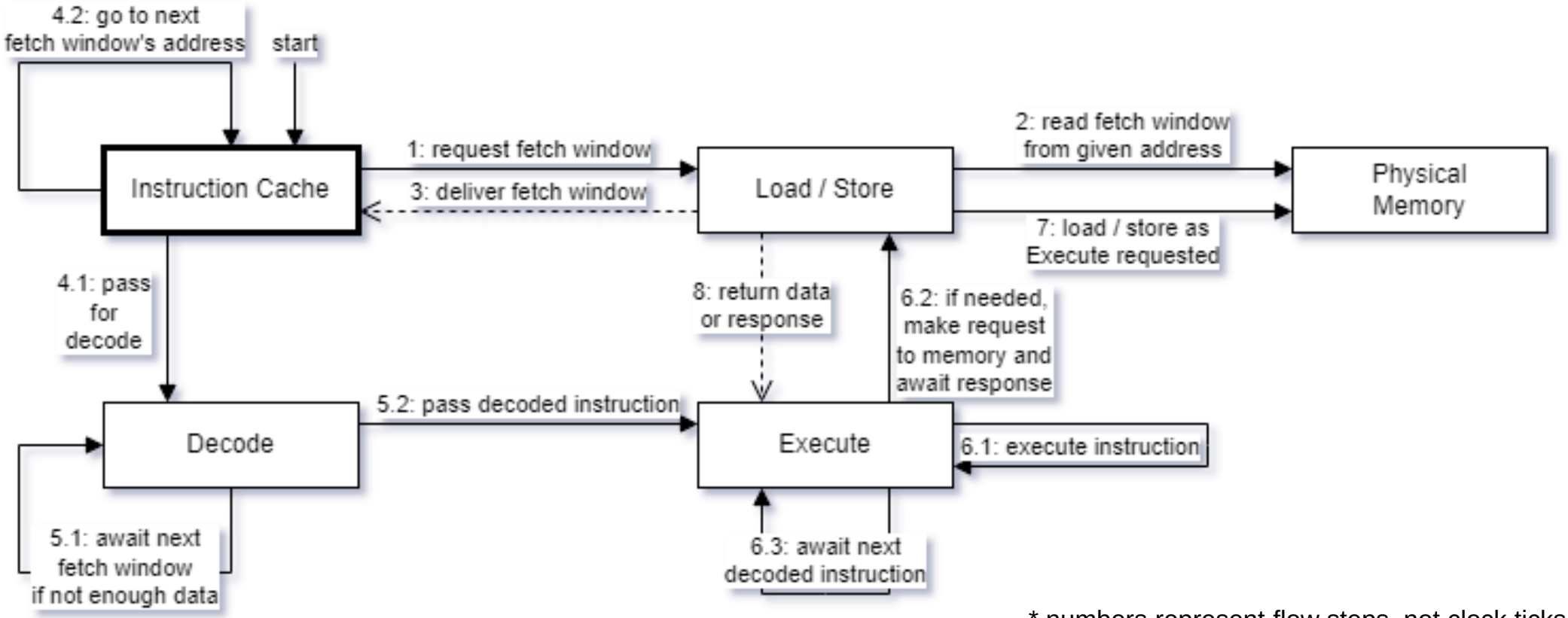
Output options:

- To console
- To output file
- To output file + Memory dump file

Logging: module's name + timestamp + info about:

- Request received / sent
- Operation began / finished
- Signals received / sent
- Discards
- Exceptions encountered
- Other module-specific info

Collaboration between threads during regular execution



* numbers represent flow steps, not clock ticks

The **Instruction Cache** – starts & sustains the simulation, constantly feeding fetch windows to DE

Execute – ends the simulation when either `**end_sim**` or too many exceptions are encountered.

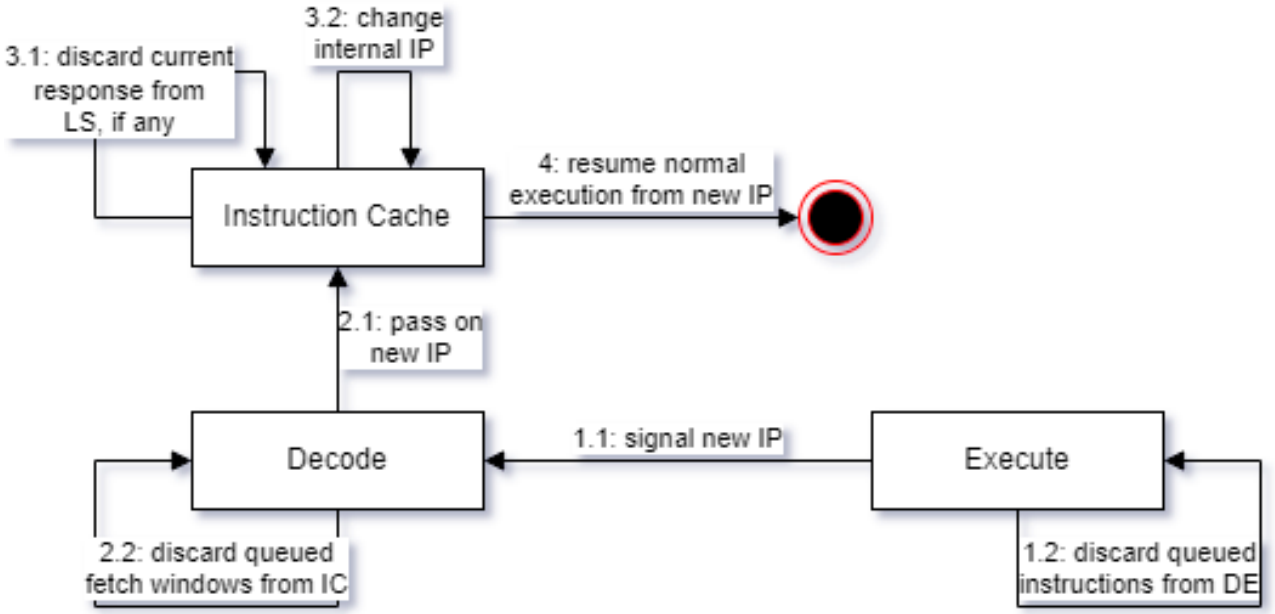
Propagation of control flow change between modules

Irregular **Instruction Pointer** changes triggered by:

- successful jump instructions
- function calls & returns
- exceptions being triggered

EX – begins the **IP** change signal chain.

These 3 modules clean their input queues of leftover data (from incremental **IP** after instruction that triggered the **IP** change).



The **IP** change Pipeline:

$$\text{EX} \xrightarrow{new\ IP} \text{DE} \xrightarrow{new\ IP} \text{IC}$$ (aligns internal IP to new value)

System Stability: Exceptions

```
10      call 555
```

```
[EX@T=688]> Encountered exception 'IP not aligned to 16b' at #022b
```

```
.vector_8
dw misaligned_ip_handler
```

```
[EX@T=728]> Calling exception handler at #1048
```

```
32    misaligned_ip_handler:
33        mov [exceptions_count], 5
34        div r4, 0
```

```
[EX@T=772]> Began executing 'div r4,  0' from #104e
[EX@T=772]> Encountered exception 'Division by 0' at #104e
    !EX forcefully ends simulation at T=772 due to double exception!
```

.vector_X – labels for exception handlers.

Possible exceptions:

- Division by 0
- Invalid source code: unknown instruction, incompatible arguments' number / types
- Load / Store request to address not aligned to 16bits
- IP not aligned to 16bits
- Stack overflow (lower or upper limit)

If handler provided for a certain exception ⟶ the simulation will call it to remedy the exception.

Two yet-untreated exceptions at the same time ⟶ simulation shuts down.

System Performance: Caching

```
[IC@T=37]> Requested fetch window starting at #1000 from LS
[LS@T=38]> Accepted request from IC regarding #1000
[LS@T=52]> Fetched [1841 1064 0000 2420] from #1000 for IC
[IC@T=54]> Delivered [1841 1064 0000 2420] from #1000 to DE
```

18 ticks vs. 2 ticks IC delivery

```
[IC@T=168]> Received signal to change IP to #1000 (aligned as #1000)
[IC@T=170]> Delivered [1841 1064 0000 2420] from #1000 to DE
    (From IC's cache)
```

```
[LS@T=123]> Accepted request from EX regarding #1064
[LS@T=129]> Fetched { 0005 } from #1064 for EX
    (Entirely using LS's cache)
```

7 ticks LS delivery

```
 [LS@T=151]> Stored { 3c00 } at #1004 for EX
     (Entirely using LS's cache)
[IC@T=169]> Invalidated cached fetch window
 from #1000 as per LS's signal
```

Maintaining code integrity

• LS - K-Set Associative Cache (64 words default) – stores words for EX;

Cache hit ⟶ 8 less ticks.

• IC - Direct-Mapping Cache (16 fetch windows default) – stores fetch windows from LS;

Cache hit ⟶ 16 less ticks: 1 itself + 15 not requesting from LS

LS – signals store operations to IC ⟶ IC invalidates cached fetch windows with those addresses

Both caches enabled ⟶ up to 25% less simulation time (depending on nature of executed code)

Used Technologies

The simulator: **C++**, using **clang**, and **cmake**.
Versioning: **git**.

Parser: **Python**.

Scripts for quick tasks & benchmark: **bash**.

Important C++ elements used & their role:

- **std::shared_ptr** for safe memory management
- **std::mutex** for avoiding race conditions & visual artifacts on output
- **std::condition_variable** for notifying CPU modules of the **Clock**'s signal
- **abstract classes** where relevant (bases for Loggers, for ExecutionStrategies within **EX** module, for the modules as clock-dependent)

Remarks, The Future of the Project, Closing Statement

While the implemented CPU model offers a reliable, informative, complex tool for running simulations with the purpose of visualizing the execution of a program at bytecode-level on the processor, there is certainly room for improvement.

Issue	Solution
Wasted cycles on jumps	A **Branch Predictor** module
Inconsistent behavior at clock periods < 1ms	Different approach to synchronization – Synchronization Barriers
Outdated, in terms of modern CPUs working with multiple Arithmetic-Logical Units for execution	Splitting **EX** into multiple **ALUs**

Continuation of development of this simulator:

I. An opportunity to develop my skills further
II. Project becoming a modern, academically-usable program

References & Webography

Pictures:

- https://stock.adobe.com/es/images/processor-cpu-microprocessor-or-chip-icon-made-with-binary-code-computer-chip-ai-chipset-digital-binary-data-and-streaming-digital-code-matrix-background-with-digits-1-0-vector-illustration/415902351 - Page 1 CPU picture
- https://www.redhat.com/sysadmin/cpu-components-functionality - Page 3 CPU Architecture picture
- https://en.wikipedia.org/wiki/C%2B%2B – Page 12 C++ logo
- https://en.wikipedia.org/wiki/CMake – Page 12 cmake logo
- https://llvm.org/Logo.html - Page 12 clang logo
- https://en.wikipedia.org/wiki/Git – Page 12 git logo
- https://en.wikipedia.org/wiki/Python_(programming_language) – Page 12 Python logo
- https://en.wikipedia.org/wiki/Bash_(Unix_shell) – Page 12 bash logo

Tools:

- Diagrams on pages 5, 8, and 9 made with draw.io: https://app.diagrams.net/

Resources:

- https://en.wikipedia.org/wiki/X86_assembly_language - for comparison of simulated language to x86 assembly
- https://www.gigabyte.com/Glossary/cisc - CISC architecture
- https://medium.com/@jaydesai36/barrier-synchronization-in-threads-3c56f947047 - Barrier Synchronization resources

Thank you.

AMD
together we advance_