Design Document - Apex CPU Simulator (Draft 3)

Structures and Functions

struct Parser

- **Fields**:
  - `filename` (char*): (of the .asm file)
  - Empty list of `struct Instruction`
- **Associated Functions**:
  - **Parse_file**: Reads the contents of the file provided. Tokenizes the content and parses them into `struct Instruction`. Stores these into the list of instructions. If a syntax error is spotted, terminates the parsing and returns an error. Otherwise, returns the list of instructions.

---

struct CPU

- **Fields**:
  - `UCRF` (struct UCRF): User control registers.
  - `UPRF` (struct UPRF): User program registers.
  - `Rename Table` (struct RenameTable): Maps architectural registers to physical registers for renaming.
  - `Forwarded Registers` (int[]): Copy of `UCRF` and `UPRF` to store forwarded register values.
  - `Forwarded Register Validity` (bool[]): Array indicating the validity of forwarded registers.
  - `Data Memory` (int[]): Array representing the data memory.
  - `Program Counter (PC)` (int): Current instruction pointer.
  - `Clock Cycles` (int): Tracks the number of clock cycles in the simulation.
  - `Code Memory` (struct Instruction[]): Array representing the program code.
  - `Stalled?` (bool): Flag indicating if the CPU is stalled due to pipeline hazards.
  - `UPRF Locks` (semaphore[]): Array of semaphores indicating the lock status of each entry in the `UPRF`.
  - `UCRF Locks` (semaphore[]): Array of semaphores indicating the lock status of each entry in the `UCRF`.
  - `IRS` (struct IRS): Queue for integer instruction reservation stations.
  - `LSQ` (struct LSQ): Queue for load/store operations.
  - `MRS` (struct MRS): Queue for multiply operations.
  - `ROB` (struct ROB): Queue for the reorder buffer managing instruction completion and commit.
  - `Fetch` (struct CPU_Stage): Represents the state of the fetch stage.
  - `D1` (struct CPU_Stage): Represents the state of the first decode stage.
  - `D2` (struct CPU_Stage): Represents the state of the second decode stage.
  - `Int FU` (struct CPU_Stage): Represents the state of the integer functional unit.
  - `MUL FU` (struct CPU_Stage): Represents the state of the multiplication functional unit.
  - `MEM FU` (struct CPU_Stage): Represents the state of the memory functional unit.

- `Commit` (struct CPU_Stage): Represents the commit stage.
- `Predictor Queue` (struct Predictor): Holds entries for control flow instructions for branch prediction.
- `Return Stack` (struct ReturnStack): Stores return addresses for function calls.

> **NOTE:**
>
> Our implementation of the APEX pipeline treats LSQ, IRS, MRS, and ROB as a separate stage in the pipeline. This is done to simplify simulating clock cycles since these structures require one clock cycle each to process.

- **Associated Functions**:
  - **Initialize**: Allocates memory for the CPU, uses `struct Parser` to parse the input file into a list of instructions and stores them in code memory.
    - **Arguments**: `filename`
    - **Returns**: Allocated CPU struct
  - **Simulate_Cycle**: Simulates one clock cycle of the CPU. In the clock cycle it will execute all the stages of the CPU by calling their respective functions. When the clock cycle simulation is complete, returns the status of the overall simulation (i.e., `true` if HALT instruction was completed or else `false`).
    - **Arguments**: `CPU`
    - **Returns**: Simulation status (`bool`)
  - **Fetch**: Using the program counter (`pc`) reads an instruction from the code memory. Additionally, checks the `struct Predictor` and `struct ReturnStack` if an entry for the current instruction address exists. If a predictor or stack entry exists, then sets the next `pc` to the value in the predictor or stack entry, or else adds `4` to the current `pc` to fetch the next instruction (in the next cycle).
    - **Arguments**: `CPU`
    - **Return**: `void`
  - **D1 (Decode 1)**: Creates an empty `struct IQE` (Instruction Queue Entry). Parses the instruction forwarded by `Fetch` into the newly created `IQE` (OPCODE, architectural register names, etc.). If the decoded instruction was either a Branch, JALP, or RET instruction, then it also creates a `struct PredictorEntry` and fills in all fields except for the predicted address. Then it adds this predictor entry to the `struct Predictor`. (The predicted address will be filled by the `IntFU` stage)
    - **Arguments**: `CPU`
    - **Return**: `void`
  - **D2 (Decode 2)**: Receives the `IQE` from the `D1` stage. Uses `struct RenameTable` to rename the architectural registers to physical registers and updates the `IQE` with the renamed registers. Then it forwards the actual `IQE` to `struct ROB` and passes a pointer to the `IQE` to either `LSQ`, `IRS`, or `MRS` depending on the type of the instruction for further processing. This

stage can stall if the `LSQ`, `IRS`, or `MRS` are at capacity. (This will also stall the previous stages) Once `LSQ`, `IRS`, or `MRS` have space to accept instructions, it will forward and stop stalling.

    - **Arguments**: `CPU`
    - **Return**: `void`

- **IRS**: In this stage, we retrieve an `IQE` from `struct IRS`. If a valid `IQE` was provided, then we forward it to `IntFU` stage. (`struct IRS` can also return No IQE if none of the stored instructions have updated register values or forwarded register values)

    - **Arguments**: `CPU`
    - **Return**: `void`

- **LSQ**: In this stage, we retrieve an `IQE` from `struct LSQ`. If a valid `IQE` was provided, then we forward it to `MemFU` stage. (`struct LSQ` can also return No IQE if none of the stored instructions have updated register values or forwarded register values)

    - **Arguments**: `CPU`
    - **Return**: `void`

- **MRS**: In this stage, we retrieve an `IQE` from `struct MRS`. If a valid `IQE` was provided, then we forward it to `MulFU` stage. (`struct MRS` can also return No IQE if none of the stored instructions have updated register values or forwarded register values)

    - **Arguments**: `CPU`
    - **Return**: `void`

- **Int FU (Integer Functional Unit)**: Retrieves an `IQE` from `IRS`. Executes integer instructions (e.g., ADD, SUB). If the instruction being processed is either a Branch, JALP, or RET instruction, then it checks if a corresponding `PredictorEntry` exists in `struct Predictor`. If the entry exists, then it will update the predicted address in the `PredictorEntry`. For Branching or Jumping instructions (including RET), if there was a miss in the prediction, it causes the CPU to flush all new instructions and resets the state of the CPU to the saved state.

    - **Arguments**: `CPU`
    - **Return**: `void`

- **MUL FU (Multiplication Functional Unit)**: Retrieves an `IQE` from `MRS`. Executes multiplication operations.

    - **Arguments**: `CPU`
    - **Return**: `void`

- **MEM FU (Memory Functional Unit)**: Retrieves an `IQE` from `LSQ`. Executes memory load/store operations.

    - **Arguments**: `CPU`
    - **Return**: `void`

- **ROB (Reorder Buffer)**: Checks if the top instruction in `struct ROB` has completed execution. If it has completed, then it is forwarded to `Commit` stage.

        - **Arguments**: `CPU`
        - **Return**: `void`

    - **Commit**: Updates the values of `struct UPRF` and `struct UCRF` of the CPU. If the current instruction was HALT, returns `true` or else returns `false`.

        - **Arguments**: `CPU`
        - **Return**: Simulation status (`bool`)

---

struct Instruction

- **Fields**:
    - `Opcode (str)` (string): String representation of the instruction's opcode (e.g., "ADD", "SUB").
    - `Opcode (int)` (int): Integer value corresponding to the opcode.
    - `Rd` (int): The destination register index.
    - `Rs1` (int): The first source register index.
    - `Rs2` (int): The second source register index.
    - `Rs3` (int): The third source register index (optional).
    - `Imm` (int): The immediate value (if applicable).

This is a simple struct only responsible for storing the parsed instruction by `struct Parser`.

---

struct UCRF

- **Fields**:

    - `registers` (int[10]): Array holding the values of user control registers.

- **Associated Functions**:

    - **Get_Register_Value**: Returns the value stored at register index `r_idx`.

        - **Arguments**: `r_idx` (The index of the physical CC register)
        - **Returns**: Value in the register.

    - **Set_Register_Value**: Stores the given value in register index `r_idx`.

        - **Arguments**: `r_idx` (The index of the physical CC register), Value to store.
        - **Returns**: `void`

---

struct UPRF

- **Fields**:

    - `registers` (int[60]): Array holding the values of user program registers.

- **Associated Functions**:

    - **Get_Register_Value**: Returns the value stored at register index `r_idx`.

        - **Arguments**: `r_idx` (The index of the physical register)
        - **Returns**: Value in the register.

    - **Set_Register_Value**: Stores the given value in register index `r_idx`.

        - **Arguments**: `r_idx` (The index of the physical register), Value to store.
        - **Returns**: `void`

---

struct RenameTable

- **Fields**:

    - `table` (int[]): Array mapping architectural registers to physical registers for renaming.
    - `free_list` (int[]): Array containing the free physical registers.

- **Associated Functions**:

    - **GetMapping**: Returns the currently mapped physical register for the given architectural register.

        - **Arguments**: Index of architectural register
        - **Returns**: Index of physical register

    - **RenameRegister**: Renames the given architectural register to a physical register from the free list. Also frees the physical register previously mapped to the given architectural register. Uses the free list as a FIFO queue.

        - **Arguments**: Index of architectural register
        - **Returns**: Index of renamed physical register

---

struct IQE

Fields:

- `Opcode` (int): Integer value representing the instruction's opcode.
- `Rd` (int): The destination register.
- `Rs1, Rs2, Rs3` (int): Source registers.
- `Imm` (int): Immediate value.
- `Rd_value, Rs1_value, Rs2_value, Rs3_value` (int): Actual values of the registers.
- `Completed` (bool): Indicates if the instruction has completed execution.
- `Rename Table` (int[]): Current state of the register renaming table.
- `Free List` (int[]): Current state of the free list of registers.
- `Forwarded Registers` (int[]): Current state of the forwarded register file.
- `Forwarded Register Locks` (bool[]): Current state of the forwarded register locks.
- `Predictor Queue` (struct PQE[]): Current state of the branch predictor queue.
- `Return Stack` (struct RSE[]): Current state of the return stack.

This is a simple struct only responsible for storing the instructions in the pipeline.

---

struct IRS

- **Fields**:
  - `queue` (IQE[8]): A list of instructions stored in the IRS.
- **Associated Functions**:
  - **GetInstruction**: Checks for an instruction in the queue that has all values fulfilled by UPRF/UCRF or forwarded. Returns an instruction if it has all available values or else returns NULL.
    - **Arguments**: `void`
    - **Returns**: IQE with all valid register values

---

struct LSQ

- **Fields**:
  - `queue` (IQE[6]): A list of instructions stored in the LSQ.
- **Associated Functions**:
  - **GetInstruction**: Checks for an instruction in the queue that has all values fulfilled by UPRF/UCRF or forwarded. Returns an instruction if it has all available values or else returns NULL.
    - **Arguments**: `void`
    - **Returns**: IQE with all valid register values

---

struct MRS

- **Fields**:
  - `queue` (IQE[2]): A list of instructions stored in the MRS.
- **Associated Functions**:
  - **GetInstruction**: Checks for an instruction in the queue that has all values fulfilled by UPRF/UCRF or forwarded. Returns an instruction if it has all available values or else returns NULL.
    - **Arguments**: `void`
    - **Returns**: IQE with all valid register values

---

struct PQE (Prediction Queue Entry)

Fields:

- `PC` (int): The address of the control flow instruction.
- `Type` (string): The type of control flow instruction (e.g., Branch, JALP, RET).
- `Next_PC` (int): The predicted next instruction address.

This is a simple struct only responsible for storing the parsed instruction by `struct Predictor`.

---

struct RSE (Return Stack Entry)

Fields:

- `Return Address` (int): The address to return to after a function call.

This is a simple struct only responsible for storing the parsed instruction by `struct Predictor`.

---

struct Predictor

- **Fields**:
  - `Predictor Queue` (PQE[]): Holds entries for control flow instructions for branch prediction.
- **Associated Functions**:
  - **GetPredictorEntry**: Checks if the provided instruction address has a value in the predictor queue. If an entry exists, then returns the stored predicted address.
    - **Arguments**: Instruction Address
    - **Returns**: Predicted Address
  - **UpdatePredictorEntry**: Updates the predicted address in the predictor queue for the instruction address provided.
    - **Arguments**: Instruction Address, Predicted Address
    - **Returns**: `void`

---

struct ReturnStack

- **Fields**:
  - `Return Stack` (RSE[]): Stores return addresses for function calls.
- **Associated Functions**:
  - **GetReturnAddress**: Returns the top entry in the return stack.
    - **Arguments**: `void`
    - **Returns**: Return Address
  - **AddReturnAddress**: Pushes the given return address to the top of the stack.
    - **Arguments**: Return Address
    - **Returns**: `void`

---

struct CPU_Stage

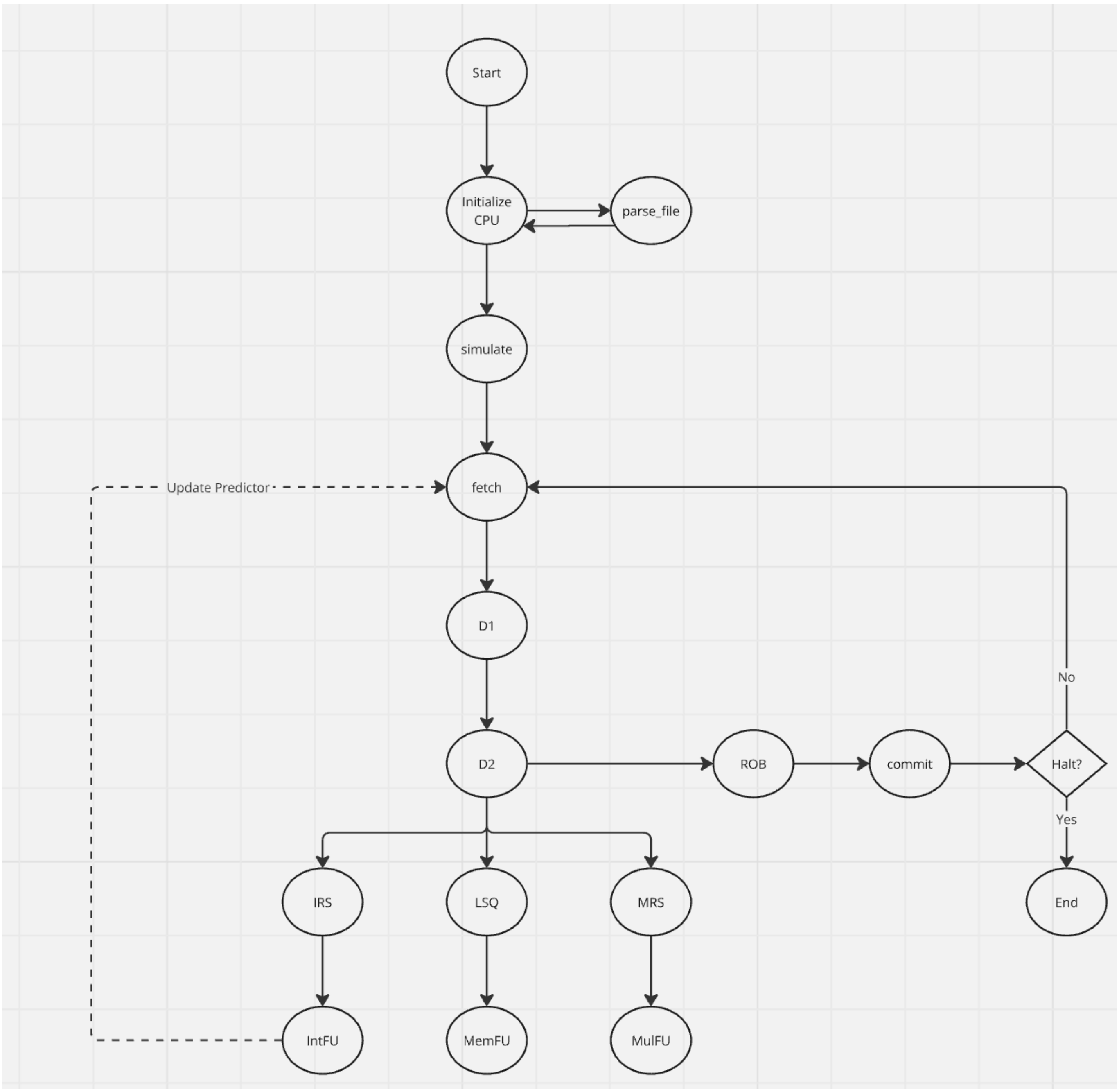
Fields:

- `Has Instruction` (bool): Flag indicating if there is an instruction in the stage.
- `Entry` (IQE): The instruction entry currently in the pipeline stage.

This is a simple struct only responsible for storing the values in a stage in the APEX pipeline.

Flow diagram of functions

> **NOTE:**
>
> This is a simple flow diagram of the pipeline to show the order of the functions being called. Details have been omitted to keep this diagram simple. A detailed explanation of all functions and how they work are given above.


Start
Initialize CPU
parse_file
simulate
Update Predictor
fetch
D1
No
D2
ROB
commit
Halt?
Yes
IRS
LSQ
MRS
End
IntFU
MemFU
MulFU