Design Document

cpu_emu

(In Development)

Joseph MacDonald

Independent Project

Contents

Table of Contents

Section 1 - ISA Design

1. Instruction and Data Design

a. Notes

Instr/Data Bit of 1 indicates presence of an instruction

Reg/Literal Bit of 1 indicates the presence of a register reference

All instructions and data are stored in 64-bit chunks

Memory is referenced using 32-bit addresses

All 64-bit data values are stored in Two's Complement form

ASCII-bit indicates whether data should be output in an ASCII form

b. Instruction Types

Type	Format	Description	Examples

ADD	Type 1	Performs addition on two input values, stores in specified register. Can take either register or literal operands	ADD R0, R1, R2
			ADD R1, R1, #10
			ADD R10, #15, #23

SUB		Subtracts second value from first, stores in specified register. Can take either register or literal values as operands	SUB R0, R5, R4
			SUB R0, #100, R7
			SUB R9, R6, #10

MULT		Multiplies two values together, stores in specified register. Can take register or literal operands. In effect, repeat addition for specified amount	MULT R7, R4, R2
			MULT R4, #42, R9
			MULT R4, R6, #9

AND		Performs a bitwise AND operation on the two input values. Stores in specified register, can take register or literal operands	AND R0, R2, R9 AND R3, #12, R9 AND R5, #12, #34
OR		Performs a bitwise OR operation on the two input values. Stores in specified register, can take register or literal operands	OR R3, R3, R4 OR R5, #92, #100 OR R7, #12, R1
XOR		Performs a bitwise XOR operation on the two input values. Stores in specified register, can take register or literal operands	OR R6, R11, R2 OR R12, #99, R1 OR R7, #104, R4
NOT	Type 2	Performs a bitwise NOT operation on the input. Stores in specified register, can take a register or literal operand	NOT R1, R1 NOT R10, #2
FLIP		Flips the sign of the input, be that literal or register	FLIP R4, R8
STR	Type 3	Stores specified register contents at the memory address supplied	STR R2, &1223ABCD STR R0, &A1B2C3D4
LDR		Loads the value stored at the specified memory address into the register supplied	LDR R5, &AEEC1442 LDR R8, &12345678

CMP	Type 4	Compares two input values, setting ALU flags but not changing the value in any register. Can take literal or register inputs	CMP R4, R7
			CMP #9, R2
			CMP R10, #0

B	Type 5	Performs a branch operation, setting the PC to the position specified address if any conditions are met. Find branch conditions in table 'c. Branch Conditions'	B &ABCD1234 BNE &12344321 BLT &FFECFDDA
HLT	Type 6	Stops the clock, ending any running operations	HLT
OUT	Type 7	Outputs the value in the specified register to an external file. Will output in ASCII form if the 'A' flag is set	OUT R R7 OUT A R1

c. Instruction Formats

Type 1 – Arithmetic & 2-Operand Bitwise Operations

[Instr Type]{4} [Return Register]{4} [Reg/Literal Bit 1] [Reg Reference/Literal Value]{16} [Reg/Literal Bit 2] [Reg Reference/Literal Value]{16} [Padding]{20}

Type 2 – Single-Operand Bitwise Operations

[Instr Type]{4} [Return Register]{4} [Reg/Literal Bit 1] [Reg Reference/Literal Value]{16} [Padding]{40}

Type 3 – Memory Operations

[Instr Type]{4} [Referenced Register]{4} [Memory Address Reference]{32} [Padding]{24}

Type 4 – Comparator Operations

[Instr Type]{4} [Reg/Literal Bit 1] [Reg Reference/Literal Value]{16} [Reg/Literal Bit 2] [Reg Reference/Literal Value]{16} [Padding]{24}

Type 5 – Branch Operations

[Instr Type]{4} [Branch Condition]{4} [Memory Address Reference]{32} [Padding]{24}

Type 6 – Control Operations

[Instr Type]{4} [Padding]{60}

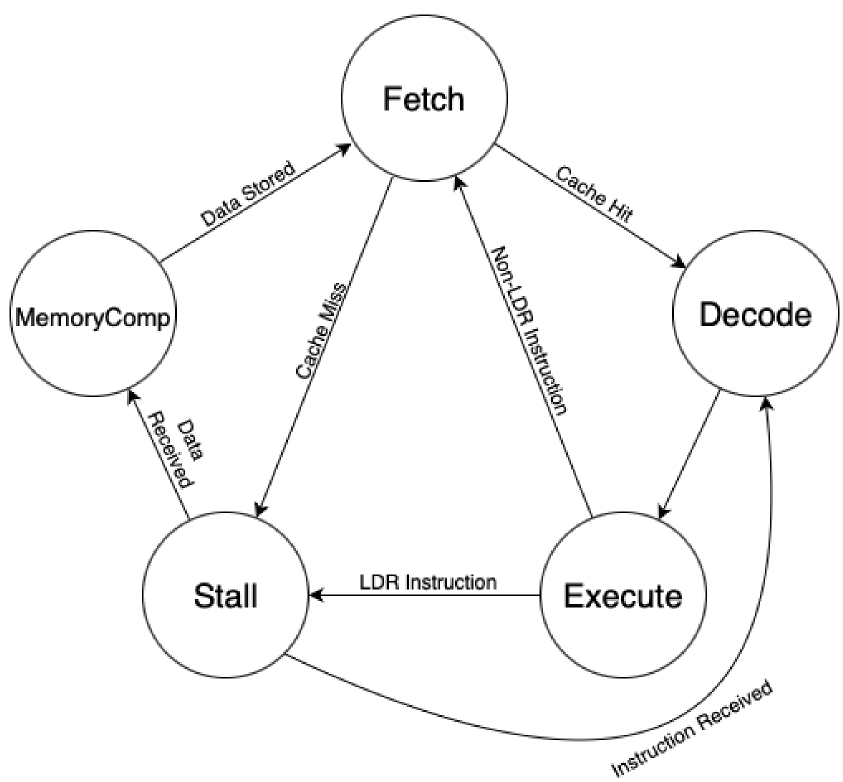
Type 7 – Output Operations

[Instr Type]{4} [Target Register]{4} [ASCII Bit] [Padding]{54}

Section 2 – Emulated System Design

1. Finite State Machine Design

State	No.	Description/Steps	Next State(s)
Fetch	0	Retrieve next instruction from memory Call cache ➔ If cache hit – Decode Instruction ➔ If cache miss – Stall, wait for RAM	Decode Stall
Decode	1	Decodes 64-bit instruction from Fetch/Stall Parses into parsedInstruction struct	Execute
Execute	2	Execute instruction from Decode In most cases, move to Fetch In case of LDR, move to Stall until the data returns from RAM	Fetch Stall
Stall	3	Wait for either instruction fetch or LDR instruction to return value from RAM Store returned instruction in MemoryInstructionRegister, ready for decoding Move to MemoryComp if data returned, store in Memory Data Register	Decode MemoryComp
MemoryComp	4	Takes data stored in Memory Data Register, stores in the appropriate register in the CPU	Fetch

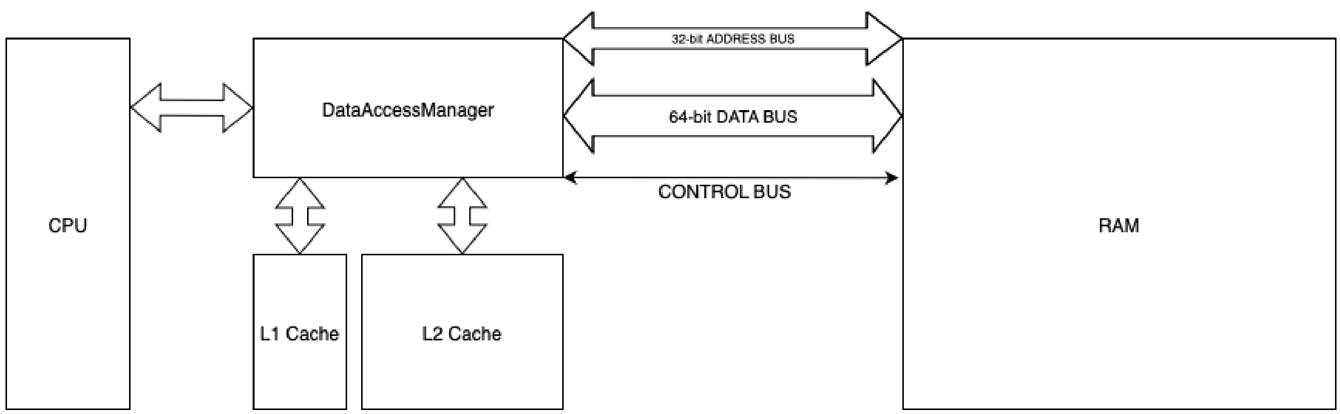


2. Data Handling Design

To truly emulate the CPU/Cache/RAM interaction, a new CPU component, the DataAccessManager, is used as a middleman to manage components and place data and addresses in the appropriate positions. The CPU itself only ever interacts with the DAM. This does mean that, in a slight break from reality, the L1 Cache is not built into the CPU object itself, but this was thought to be necessary to keep the structure as simple, readable and efficient as possible. The DAM handles two levels of cache, L1 and L2, each of different sizes, and with L1 having priority. The DAM organises dataflow between the caches, the CPU, and the memory buses, and allows for cascading data to flow from L1 into L2 in the event of an overwrite in L1.

The DAM also manages memory interaction.

Both cache modules are built to a LRU format, using a queue to keep the most relevant data.



IPC Data Protocol – WORK-IN-PROGRESS

Dashboard to Emulator Messages

Message	String Format
Get Memory Data from Location	GET//<Memory Address>
Increase Clock Speed	INC_CLK
Decrease Clock Speed	DEC_CLK
Set Clock Speed	SET_CLK//<Val>

Emulator to Dashboard Messages

Message Type	String Format
Update Register Value	REG//<Reg Num>//<Val>
Update State Value	STATE//<State No.>
Update Instruction Values	INSTR//<Current Instr Hex>//<Incremented Instr Hex>
Return Memory Data	RAM//<Memory Address>//<Memory Data>
L1 Cache Utilisation	L1_UTIL//<Val>
L2 Cache Utilisation	L2_UTIL//<Val>