DECA Lab Spring Term

Spring Part 1: EEP1 Datapath & ALU instructions

Department of Electrical and Electronic Engineering

Imperial College London

v2.0

Spring 2025

Contents

Introduction

You are expected to complete Lab1 (this handout) in two weeks.

Before the lab

- Check from the Issie release page that you have the latest version - if not download it.
- From Spring, Lab1 in the DECA github repository Download the `Lab1-2025` directory.
- Follow Section 1 below to set up the EEP1 assembler on your own laptop, if you have not already done so in class 1
- Check, from Lecture 2, that you understand what each of the inputs and outputs of the `eep1lab1` `reg16x8` sheet do, and what type of register file this sheet implements.

In this lab you will:

i. Learn to view waveforms in the Issie waveform simulator

ii. See how ALU instructions are implemented by executing these instructions using EEP1 CPU in the Issie project `eep1lab1`.

iii. Analyse the datapath hardware in the `eep1lab1` EEP1 CPU to determine how it works.

iv. Use algebraic truth-tables to describe how data flows through multiplexers and design control logic.

Do note that some of the jump instructions in the hardware of `eep1lab1` EEP1 CPU are not complete. You will add hardware to implement these in a later lab focusing on the CPU control path. All of this lab can be completed without changing the hardware in `eep1lab1`.

1 The EEP1 assembler

□ **Task 1.** *Follow the instructions in the Introduction to download* `eep1lab1` *Issie project, and update Issie.*

The EEP1 assembler tool `eepassembler` converts EEP1 assembly code into machine code. If you have not already done so set up `eepassembler` so that you can convert an assembly code `.txt` file in an Issie project into machine code as follows:

1. Download and unzip the `eepassembler` software source (or fork and clone the github repo).
2. Put `eepassembler` and `eep1lab1` folders in the same parent folder. Start a terminal (Windows cmd) in the eepassembler root directory that contains the README. Then, for example, the file path to run the assembler on `*.txt` assembly files in the `eep1lab1` directory, as used by the Issie `eep1lab1` project, might be `dotnet run ../eep1lab1` (Windows `dotnet run ..\eep1lab1`). If you do not follow this you will need to work out the path yourself - or on windows use `chooser.bat` to select the Issie directory and run eepassembler automatically.
3. Follow the README instructions to run the assembler program, and check that it generates four `.ram` files from the four `.txt` files in your eep1lab1 project.

2 Data flows in EEP1 ALU instructions

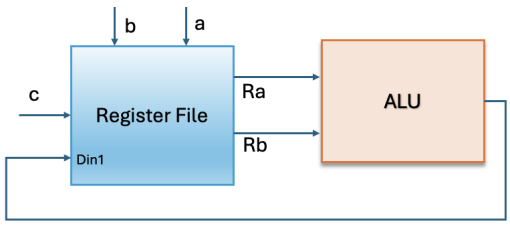

Figure 1: EEP1 basic datapath

CODEMEM memory address	Assembler	CODEMEM Data	Operation
0x0000	`MOV R3, #64`	`0b0000,0111,0100,0000`	`R3 := 64`
0x0001	`MOV R5, R3`	`0b0000,1010,0110,0000`	`R5 := R3 // 64`
0x0002	`ADD R4, R5, R3`	`0b0001,1010,0111,0000`	`R4 := R5 + R3`
0x0003		`0`	

Figure 2: Instruction Memory Data

Before we consider in depth how each ALU instruction works, we must first understand how the ALU values are generated from the instructions. [Note: You should only begin this portion after doing Problem Sheet 1, Q1.1 - Q1.4]

□ **Task 2.** *Open* `lab1testsimple.txt` *file. Refer to the ALU instruction model in Figure 9.*

1. *The ALU block port input names show which register (a or b) in the instruction is which input. Work out the required register file addresses AD1, AD2, AD3, in binary, for each given instruction as shown in Figure 2. Check your answers against Issie. Note which of these addresses are not required (don't care) for the MOV instructions.*
2. *Figure 1 shows a simplified diagram of how data flows in the EEP1 datapath. Look at the datapath sheet and the ALU sheet in eep1lab1 and trace the paths the 16 bit data for each instruction flows between input and output component ports in these sheets, listing in order all of the Issie components the data flows through.*
3. *Add in the components in reg16x8, so you are tracing 16 bit data flows between register Q outputs and D inputs. This will consolidate your understanding of Lecture 2.*

   [Hint: For the datapath sheet, trace data between the 16 bit inputs and outputs of the Register File (REG16x8), the IMMEXT output of EXT, and the ALU. For the ALU sheet, look at all of the 16 bit inputs and how data flows through the output.]

3 EEP1 ADD & MOV Instructions: Timing

You can simulate the EEP1 CPU (`eep1lab1` project) with machine code instructions coming from the `lab1testmovadd.r` file as follows:

1. Open `lab1testmovadd.ram`, and `lab1testmovadd.txt`. Note that each line of the assembler file generates one 16 bit hex word in the machine code file as specified in Figure 9.
2. Start Issie. Open the `eep1lab1` project you downloaded previously. Open the `eep1` sheet. Select the `Codemem` ROM properties and check the ROM is linked to `lab1testmovadd.ram`. If not change the linking.
3. Check that the Codemem ROM has the same data shown in the machine code (.ram) file by opening up the ROM viewer from properties in Issie.
4. Run the Issie waveform simulator (`Simulations->Wave Simulation` tab). Using the waveform simulator GETTING STARTED button for help if needed, you can adjust the display to make waveforms as in Figure 11. Check that you can do the following:
   - Alter the order of displayed waveforms by dragging.
   - Delete a displayed waveform
   - Add a new displayed waveform by searching for part of its name on the Select Waves screen
   - Change the radix of displayed waveforms
   - Change the position of the cursor by clicking the waveforms.
   - Adjust the grey vertical divider to make the simulation tab with waveforms wider.
   - Adjust horizontal scale so that you can view the first 8 cycles of the CPU operation while also seeing the value of the PC (`PCV`) displayed in hex.

□ **Task 3.** *Use Issie's waveform simulator to view the first 8 clock cycles of sheet `eep1` simulation using the `lab1testmovadd.txt` instructions.*

i. Determine in which clock cycle each instruction controls the data in the datapath and ALU.

ii. Determine when each register output changes in relation to the ALU data that is written.

iii. For each of the registers `REGn.Q` on sheet reg16x8 explain what causes every change and the changed value..

For your convenience these outputs are connected to Issie viewer components `R0-R7` and so visible in the Step Simulator. Use the instructions in `lab1textmovadd.txt` and the ALU instruction reference in Figure 12.

□ **Task 4.** *For both MOV and ADD instructions with literal and register op2 (4 cases) choose an example from `lab1testmovadd.txt` and put the instruction into a new four instruction file `lab1test4cases.txt`. Use `eepassembler` to create `lab1test4cases.ram`. Use Figure 9 to calculate the corresponding machine code in binary, then convert it to hexadecimal. Check that your work is in agreement with `lab1test4cases.ram`.*

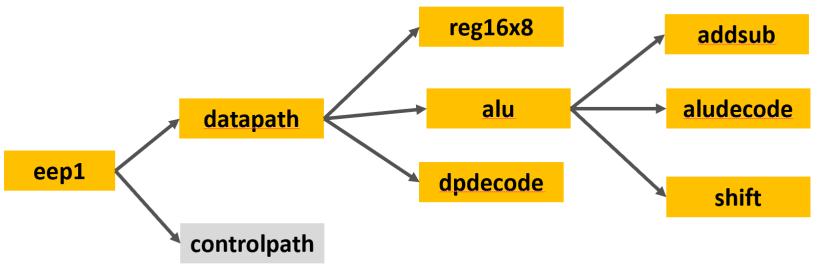

Figure 3: EEP1 datapath design hierarchy

Block	Select	Use
Datapath.MUX1	AD1SelC	determines REGFILE Port 2 address.
Datapath.MUX2	0	Non-zero for memory load - assume 0 in Lab1.
Datapath.MUX3	0	Non-zero for writing PC to registers (lab2) - assume 0 in Lab1.
ALU.MUX1	OP2Sel	Selects op2 format.
ALU.MUX2	ALUOPC	Selects which ALU operation.
ALU.ADDSUB	n/a	For Section 3 Assume `ADDSUB.OUT=ADDSUB.INA+ADDSUB.INB`.
EXTEND	n/a	For Lab1 assume `EXTEND.IMMEXT` = `EXTEND.IMMS8` sign extended to 16 bits.
Datapath.REGFILE	n/a	See lecture notes slide 37 for truth table of `DOUT` as function of `AD` for each port.

Figure 4: Data flow through datapath block

Algebraic Truth Tables for Dataflow in the EEP1 Datapath

Figure 5 shows the top-level schematic sheet of the EEP1 CPU in Issie, and Figure 3 the design hierarchy of its `datapath` sheet. This section will work with the EEP1 datapath hardware in the `eep1lab1` project. Specifically, you will trace the logic through the `datapath` and `alu` sheets and work out the required control signals to correctly change a register file register as required by the MOV and ADD instructions.

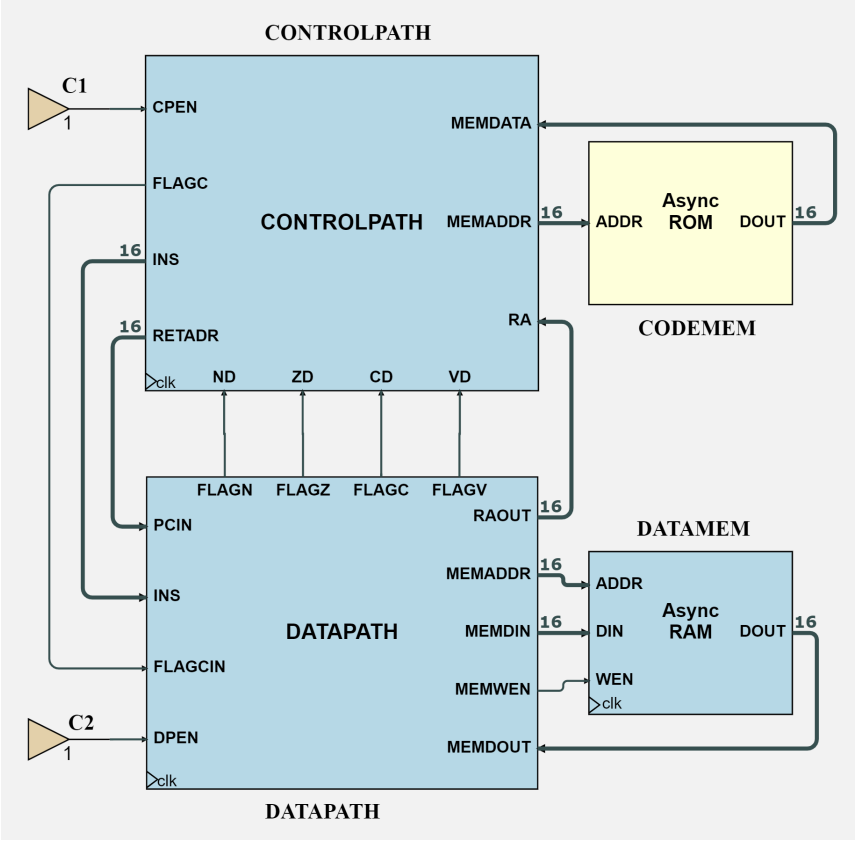

Figure 5: EEP1 CPU

Look at the `datapath` sheet (Figure 6) in Issie. Note that the `sheets` menu shows. The current instruction word (that is - the machine code for the current instruction) is output from the Instruction ROM and input to `datapath` on its `INS(15:0)` port. At every clock cycle, a new instruction will be presented. The function of the datapath is to change one of the CPU registers contained in `REGFILE` as specified by this instruction. The instruction word INS(15:0) is interpreted by the `dpdecode` sheet which contains combinational logic that drives control signals in the datapath correctly to implement every instruction.

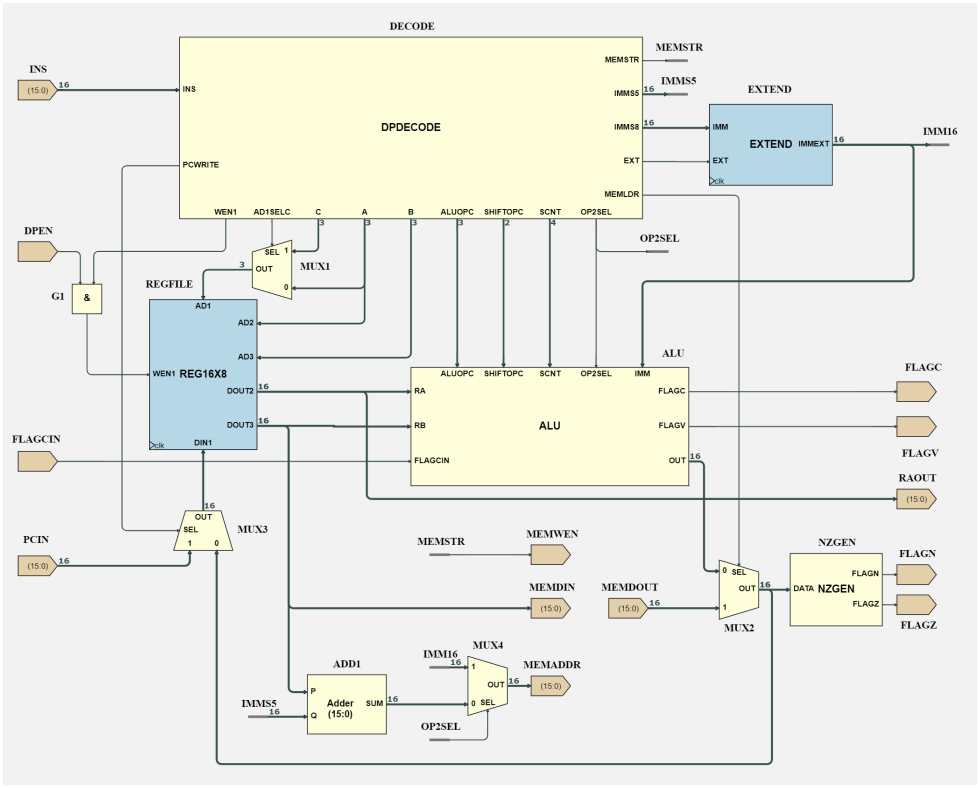

Figure 6: EEP1 Datapath

Figure 6 shows the 16 bit data flow though the datapath from `REGFILE.` read port(s), through hardware blocks, to the `REGFILE` write port `Regfile.DIN1`. Figure 4 explains the function of the MUXes and other blocks in this path. The flow of data is controlled by the `INS(15:0)` instruction word fields `A,B,C,Imm8` that determine which registers (or number) are operands.

Throughout the following sections, when you are asked to write algebraic truth tables, it is expected that you will work these out yourself, one row at a time, rather than use Issie truth tables. You are also expected to substitute don't cares on row inputs or outputs when operation does not depend on them. For example, the D input of a register is don't care for a row where its enable is 0.

□ **Task 5.** *Run the* `eep1` *sheet of* `eep1lab1` *with ROM contents from your* `lab1test4cases.ram` *file. For each of the 4 cases determine the values of Op2Sel, AD1SelC from the waveform simulator. Use this to write a 4 row truth table, with input the assembler instruction, and outputs the values of Op2Sel, AD1SelC. This shows the how these signals are driven by EEP1 instructions.*

Analysing the hardware in the `eep1, alu` *sheets by showing how the register file write port is controlled.*

1. *Modify the template shown in Figure 7 to make an algebraic truth table for the case that ALUOPC=0 (MOV instruction).*
2. *Explain briefly what AD1SelC and Op2Sel do.*
3. *Similarly make an algebraic table for the case that ALUOPC = 1 (ADD instruction).*

Now use your algebraic truth tables to check that the EEP1 datapath works correctly for ADD and MOV instructions. For each instruction in `lab1test4cases.txt`*, using the values of Op2Sel, AD1Sel from your simulation to select the algebraic table row, check that the algebraic truth table values of* `REGFILE.DIN, REGFILE.AD1` *implement the register file write specified in Figure 12.*

A	B	C	IMM	OP2SEL	AD1SEL	REGFILE.Din1	REGFILE.AD1
a	b	c	n	0	0		
a	b	c	n	0	1		
a	b	c	n	1	0		
a	b	c	n	1	1		

- Assume that in the Datapath IMM = IMMS8 sign extended to 16 bits
- Use notation REG[a] for REGFILE.REGa.Q (the output of register a)

Figure 7: Template for algebraic Truth-Table showing EEP1 Datapath Register Write functionality

4 ADDSUB Block design in the ALU

Look at the ALU sheet. The `ALU.ADDSUB` block is responsible for implementing addition and subtraction: its operation depends on the INVERT and ADDSUBCIN signals which come from the `ALU.DECODE` block. This block is controlled by `ALUOPC` and `FLAGCIN`. In this section you will:

- Check how the ALU logic uses one adder to implement addition and subtraction
- Relate that logic to the content of lecture 3
- Understand the logic in the `ALU.DECODE` block which control the ALU for instructions that require addition or subtraction.

□ **Task 6.** *The 5 instructions ADD, SUB, ADC, SBC, CMP use the `ADDSUB` block:*

1. *Check this by looking at which `ALU.MUX2` inputs are connected to `ADDSUB.OUT` and noting that `MUX2.Sel=ALUOPC`.*
2. *Write an algebraic truth table for ADDSUB.OUT as a function of algebraic inputs `INA, INB, CARRYIN` and binary input `INVERT`.*
3. *Using the truth table, and the specifications in Figure 12 of the 5 instructions, work out a truth table, using don't cares to simplify, for the `ALU.DECODE` block. This truth table has outputs `ALUDECODE.INVERT, ALUDECODE.ADDSUBCIN` and inputs `ALUDECODE.ALUOPC, ALUDECODE.FLAGCIN`. For instructions other than the 5 considered here `ADDSUB.OUT` is don't care. Explain this statement by looking at how `ADDSUB.OUT` is connected.*

□ **Task 7.** *Show that the logic given in sheet `aludecode` is compatible with your algebraic truth table. Show that `eep1lab1` is correct for all the instructions in `lab1testarith.txt` by changing instruction ROM sources (Issie properties) and comparing the register values in a simulation with Figure 12.*

5 CMP and AND instructions

□ **Task 8.** *Look at the logic in the `dpdecode` sheet that drives `DPDECODE.WEN1`. Write a truth table that shows the value of `WEN1` for each of the 8 ALU instructions. Compare this with Figure 12 to show it is correct. Look at the `MUX2` inputs in the `alu` sheet, and the coding of the "AND" instruction, to find what hardware implements it, Check its definition in Issie to make sure it is correct.*

□ **Task 9.** *Run a simulation of instructions in `lab1testandcmp.txt` and check that all the register value results in this match the instruction definitions in Figure 9.*

6 SHIFT instructions

In this part, you will learn how the hardware for shifting works in the `eep1lab1` CPU.

□ **Task 10.** *Using the `LSL,LSR,ASR,XSR` shift instruction definitions from the lectures work out what waveforms you expect from the code in `lab1testshift.txt` . Check this against `eep1lab1` simulation.*

These operations are correctly implemented in the `eep1lab1` design by the `Shift` sheet, which implements all 4 shift instructions, with control inputs `SCNT(3:0)` and `SHIFTOPC(1:0)`. `SCNT` controls the number of bits shifted and `SHIFTOPC` the shift type as specified in Figure 9. The combinational shift logic is implemented by the 4 shift blocks `Shift1, Shift2, Shift4, Shift8`. `Shift`n implements `SHIFT`n`.OUT`= (`SHIFT`n`.IN` shifted by x) where $x = n$ if `EN=1`, or $x = 0$ (e.g. no shift) if `EN=0`. The 4 `Shift`n blocks are connected in series as in shown in `shift`. You may assume that when shift blocks are in series the corresponding shift counts add.

□ **Task 11.** *By inspecting the Issie hardware driving `SHIFT`n`.EN`, or by noting the values of these signals for different values of `SCNT`, explain how the correct overall shift is implemented for all values of `SCNT`.*

This design, implementing a variable shift of up to 2^n bits in n stages, is one common implementation of a *barrel shifter* and is often used in CPUs like EEP1 or ARM that implement multi-bit shift instructions.

7 Reflections

In Lab1 you have explored nearly all of the design of the EEP1 datapath, and you will now understand how the CPU ALU instructions are implemented in hardware using a register file and an ALU using control signals decoded from the instruction word bits. Lab2 will focus entirely on the `controlpath` and show how this is implemented, and therefore how the EEP1 decides which address in instruction memory contains the next instruction to be executed. In Lab2 you will complete the EEP1 implementation of the jump instructions, and show how the Flags `FlagN,FlagZ,FlagC,FlagV` operate.

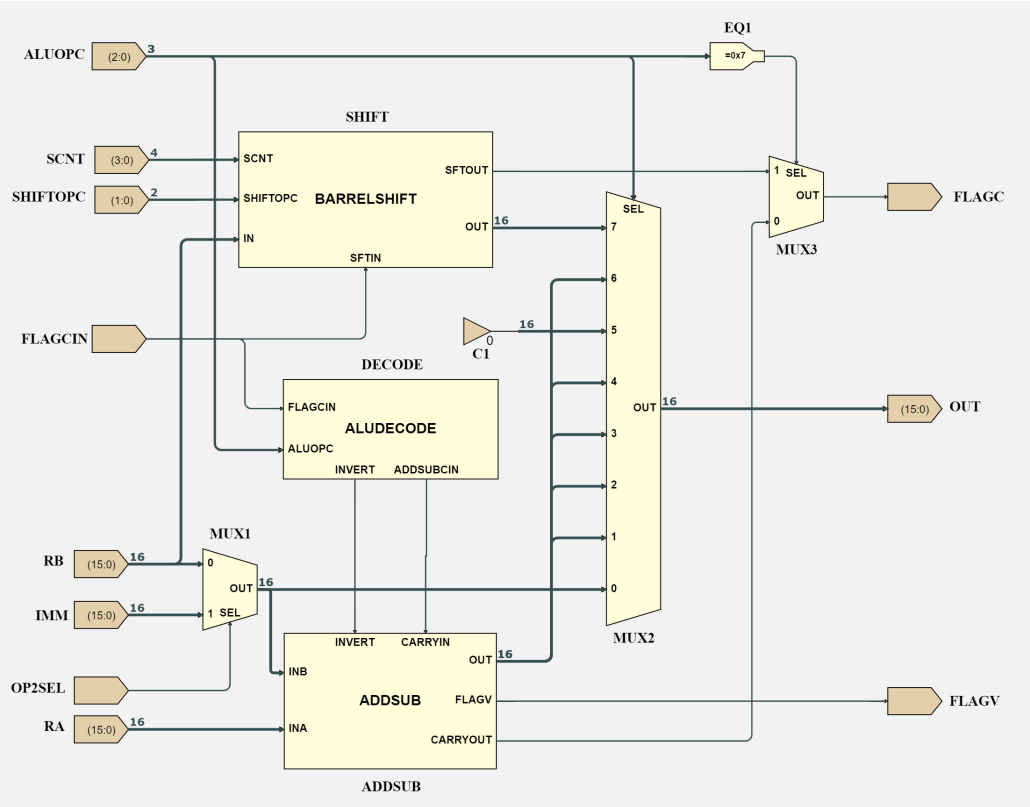

Figure 8: EEP1 ALU sheet

ALUOPC	INS(8)=0	INS(8)=1
0	MOV[1] Ra, Rb	MOV Ra, #IMM
1	ADD Rc, Ra, Rb	ADD Ra, #IMM
2	SUB Rc, Ra, Rb	SUB Ra, #IMM
3	ADC Rc, Ra, Rb	ADC Ra, #IMM
4	SBC Rc, Ra, Rb	SBC Ra, #IMM
5	AND Rc, Ra, Rb	AND Ra, #IMM
6	CMP[1] Ra, Rb	CMP Ra, #IMM
7	Shift Ra, Rb, #SCNT (see SHIFTOPC for Shift)	

[1]Instruction does not use Rc, this field is (0)

SHIFTOPC(1:0)	Shift
0	LSL
1	LSR
2	ASR
3	XSR

Legend	Meaning
Imm4	4 bit unsigned immediate
Imms8	8 bit signed immediate
(0)	Must be 0 for current instructions, non-zero values are reserved for expansion

EEP1 machine code	INS bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	ALU	0	ALUOPC=7			a			Shift-opc(1)	b			Shift-opc(0)	Imm4 (SCNT)			
			ALUOPC=0..6			a			0	b			c			(0)	
									1	Imms8 (IMM)							

Figure 9: EEP1 ALU Instructions and Machine Code

Port	Size	Type	Default	Notes	When
`INS`	16	Input		Instruction word	**This lab**
`FLAGCIN`	1	Input	0	Carry flag input	**This lab**
`FLAGC,FLAGV,FLAGN,FLAGZ`	1	Output		Flag control	Lab 2
`RAOUT`	16	Output		`RET` instruction	Lab 2
`PCIN`	16	Input		`JSR` instruction	Lab 2
`MEMDIN,MEMADDR`	16	Output		data memory interface	Lab 4
`MEMDOUT`	16	Input		data memory interface	Lab 4
`MEMWEN`	1	Output		data memory interface	Lab 4
`DPEN`	1	Input	1	Enable operation	Lab 5

Figure 10: EEP1 `datapath` sheet inputs and outputs

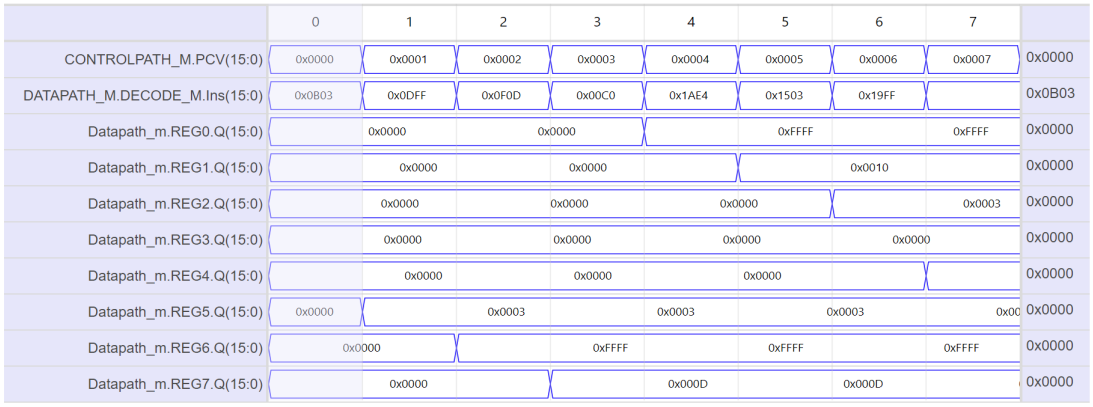

Figure 11: Simulating EEP1 running lab1testmovadd instructions

EEP1 ALU instructions: detail

OPCALU	Mnemonic	ALU.Out: INS(8)=0	ALU.Out: INS(8) = 1	Write FlagC[1]	Write register	
0	MOV	Rb	Imm	No	Ra	move
1	ADD	Ra + Rb	Ra + Imm	Yes	Rc (Ra if INS(8)=1)	arithmetic
2	SUB	Ra - Rb	Ra – Imm	Yes	Rc (Ra if INS(8)=1)	
3	ADC	Ra + Rb + C	Ra + Imm + C	Yes	Rc (Ra if INS(8)=1)	
4	SBC	Ra – Rb + (C-1)	Ra – Imm + (C-1)	Yes	Rc (Ra if INS(8)=1)	
5	AND	Ra & Rb	Ra & Imm	No	Rc (Ra if INS(8)=1)	"bitwise" logical
6	CMP	Ra - Rb	Ra - Imm	Yes	no	comparison
7	LSR, ASR, XSR, LSL	depends on SHIFTOPC		Yes[2]	Ra	shift

&	16 bit C++ bitwise AND operator

[1]**FlagC**
- 1 bit flip-flop in CPU used to store carry for multi-word addition and shifts
- Holds previous value unless written

C = FlagC.Q

[2]**Shift instructions** write FlagC with *last bit shifted out*

Figure 12: Simulating EEP1 ALU instruction detailed operation