Preparation Report LAB3

ADVANCED CPU ARCHITECTURE AND HARDWARE

ACCELERATORS LAB

361.1.4693

Introduction

This lab involves designing a simple **multi-cycle CPU** that consists of two main components:

1. **Control Unit** – implemented as a Mealy FSM, following the design principles studied in the theory course.
2. **Datapath** – executes operations in parallel according to the control signals generated by the Control Unit.

Together, these components enable the CPU to execute a set of basic instructions defined in the provided ISA:

Instruction Format	Decimal value	OPC	Instruction	Explanation	N	Z	C
R-Type	0	0000	add ra,rb,rc	R[ra]<=R[rb]+R[rc]	*	*	*
			nop	R[0]<=R[0]+R[0] (*emulated instruction*)	*	*	*
	1	0001	sub ra,rb,rc	R[ra]<=R[rb]-R[rc]	*	*	*
	2	0010	and ra,rb,rc	R[ra]<=R[rb] and R[rc]	*	*	0
	3	0011	or ra,rb,rc	R[ra]<=R[rb] or R[rc]	*	*	0
	4	0100	xor ra,rb,rc	R[ra]<=R[rb] xor R[rc]	*	*	0
	5	0101	*unused*				
	6	0110	*merge ra,rb,rc*	$R[ra]<=\{R[rb]_{MSB}, R[rc]_{LSB}\}$	*	*	0
J-Type	7	0111	jmp offset_addr	PC<=PC+1+offset_addr	-	-	-
	8	1000	jc /jhs offset_addr	If(Cflag==1) PC<=PC+1+offset_addr	-	-	-
	9	1001	jnc /jlo offset_addr	If(Cflag==0) PC<=PC+1+offset_addr	-	-	-
	10	1010	*unused*				
	11	1011	*unused*				
I-Type	12	1100	mov ra,imm	R[ra]<=imm	-	-	-
	13	1101	ld ra,imm(rb)	R[ra] <= M[imm+R[rb]]	-	-	-
	14	1110	st ra,imm(rb)	M[imm+R[rb]] <= R[ra]	-	-	-
	15	1111	done	Signals the TB to read the DTCM content	-	-	-

Note: * The status flag bit is affected , - The status flag bit is not affected

The CPU operates based on two input files and several control signals:

- **Program file (ITCMinit.txt)** – contains the program instructions in hexadecimal format, initializing the **Program Memory**.
- **Data file (DTCMinit.txt)** – contains data values for the **Data Memory**, listed in order: the first line corresponds to memory cell 0, the second line to cell 1, and so on.

The CPU also uses the following signals:

- rst – reset
- clk – clock
- ena – enable
- done – indicates when program execution is complete

At the end of execution, the CPU generates a text file containing the **final state of the Data Memory**, and the **done** flag is set to signal completion.

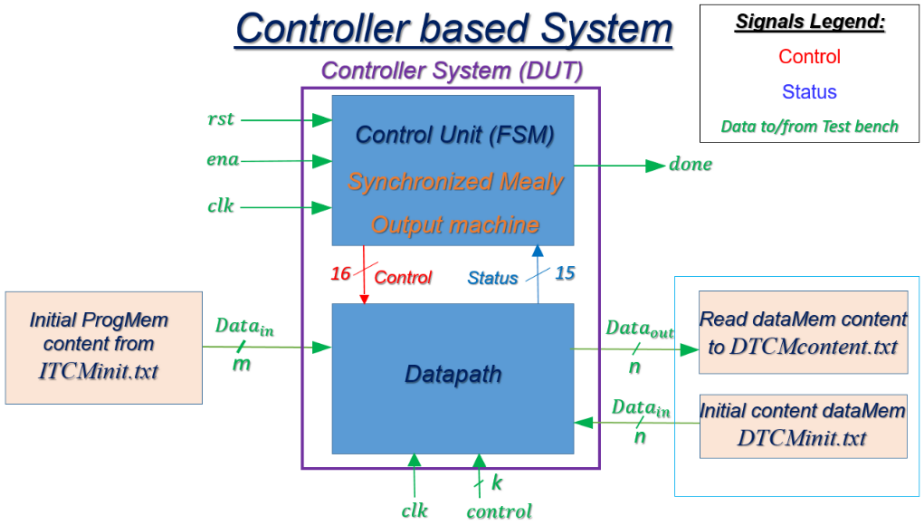
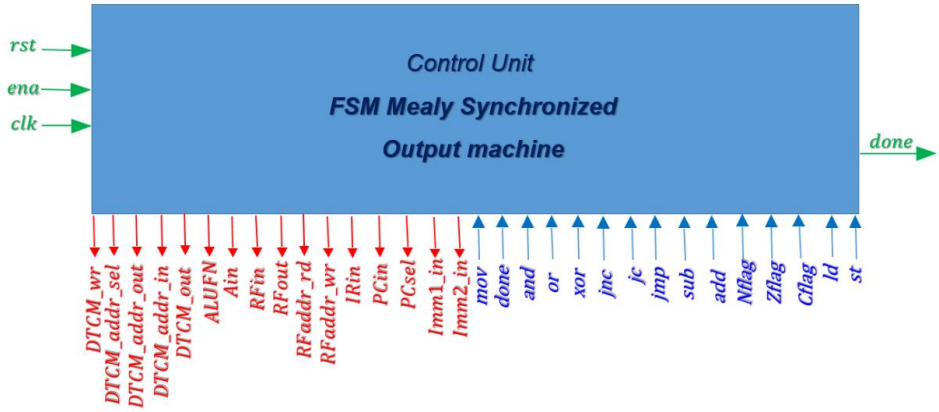


Figure 1: Overall DUT structure

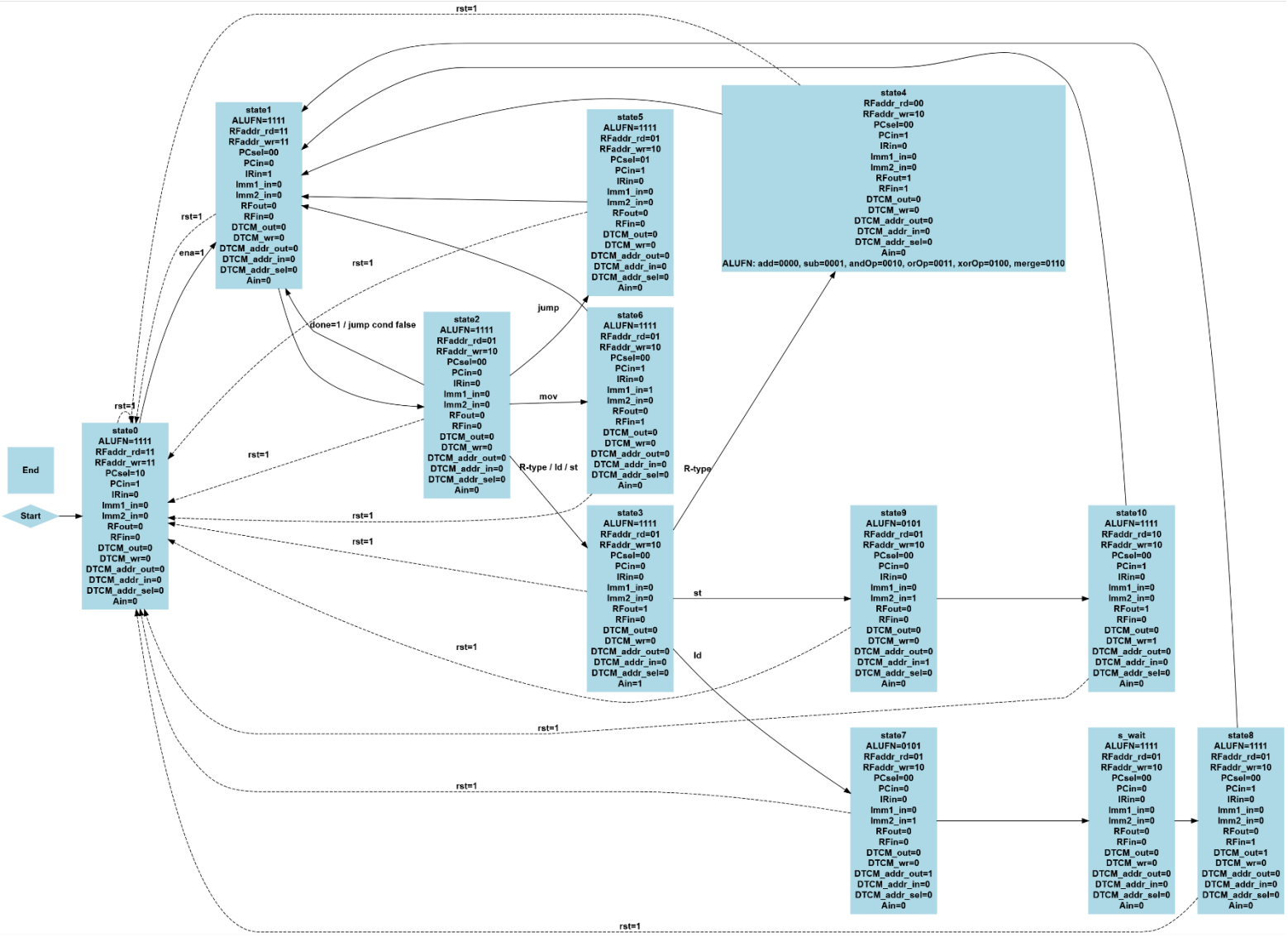
Control Unit

Control signals sent from the Control Unit to the Datapath are highlighted in **red**. Status signals returned from the Datapath to the Control Unit, which indicate the operation currently being executed after decoding the instruction from the IR, are shown in **blue**. Testbench signals (TB) are represented in **green**.



The **Control Unit** functions as the “brain” of the CPU. Its primary role is to interpret the current instruction, extracted directly from the **Instruction Register (IR)**, and generate the appropriate control signals. These signals are produced according to the design of the attached **FSM** and are sent to the **Datapath**, directing its operations.

Due to the structure of the **BUS** and the instruction set architecture (ISA), the Control Unit executes each instruction over multiple cycles. The exact sequence and timing of these cycles are specified in the following FSM diagram –

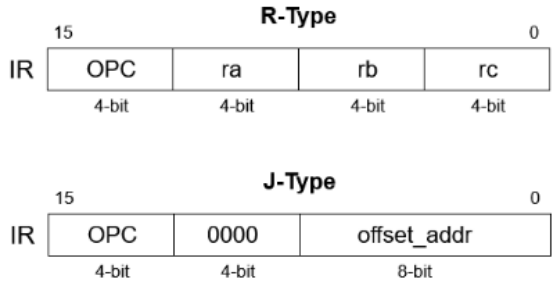


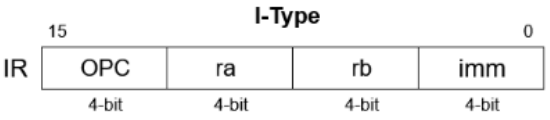
The FSM is designed to support multiple instruction types as defined in the ISA, including I-type, J-type, and R-type instructions. To organize its operation, the FSM is divided into two main groups of cycles:

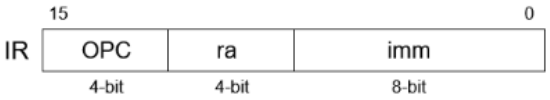
1. **Common cycles** – executed for all instructions:
    - **Fetch** – calculates the address of the next instruction and retrieves it from memory.
    - **Decode** – interprets the fetched instruction within the Datapath and informs the Control Unit, which determines the operation to execute and sets the necessary flags. This stage also retrieves any register values required by the instruction.
2. **Instruction-specific cycles** – executed based on the type of instruction:
    - **R-type** – instructions that primarily operate on registers.
    - **J-type** – instructions that involve jumps or changes in program flow.

- **I-type** – instructions that use immediate values or access memory.

This design allows the FSM to manage different instruction types efficiently while maintaining a clear separation between general instruction handling and type-specific processing. Here is the instructions formats and addressing modes:







ADDRESS MODE	SYNTAX	INSTRUCTION FORMAT	EXAMPLE	OPERATION
Register	add ra,rb,rc	R-Type	add r4,r2,r1	$R[4] \leftarrow R[2] + R[1]$
Immediate	mov ra,imm	I-Type	mov r5,0x30 mov r5,Var	$R[5] \leftarrow 0x30$ $R[5] \leftarrow Var$
Direct	ld ra,imm(r0)		ld r4,0x20(r0) ld r4, Var(r0)	$R[4] \leftarrow M[0x20]$ $R[4] \leftarrow M[Var]$
Indirect	ld ra,0(rb)		ld r4,0(r3)	$R[4] \leftarrow M[R[3]]$
Indexed	ld ra,imm(rb)		ld r4,0x20(r3) ld r4, Var(r3)	$R[4] \leftarrow M[0x20 + R[3]]$ $R[4] \leftarrow M[Var + R[3]]$

Datapath Unit

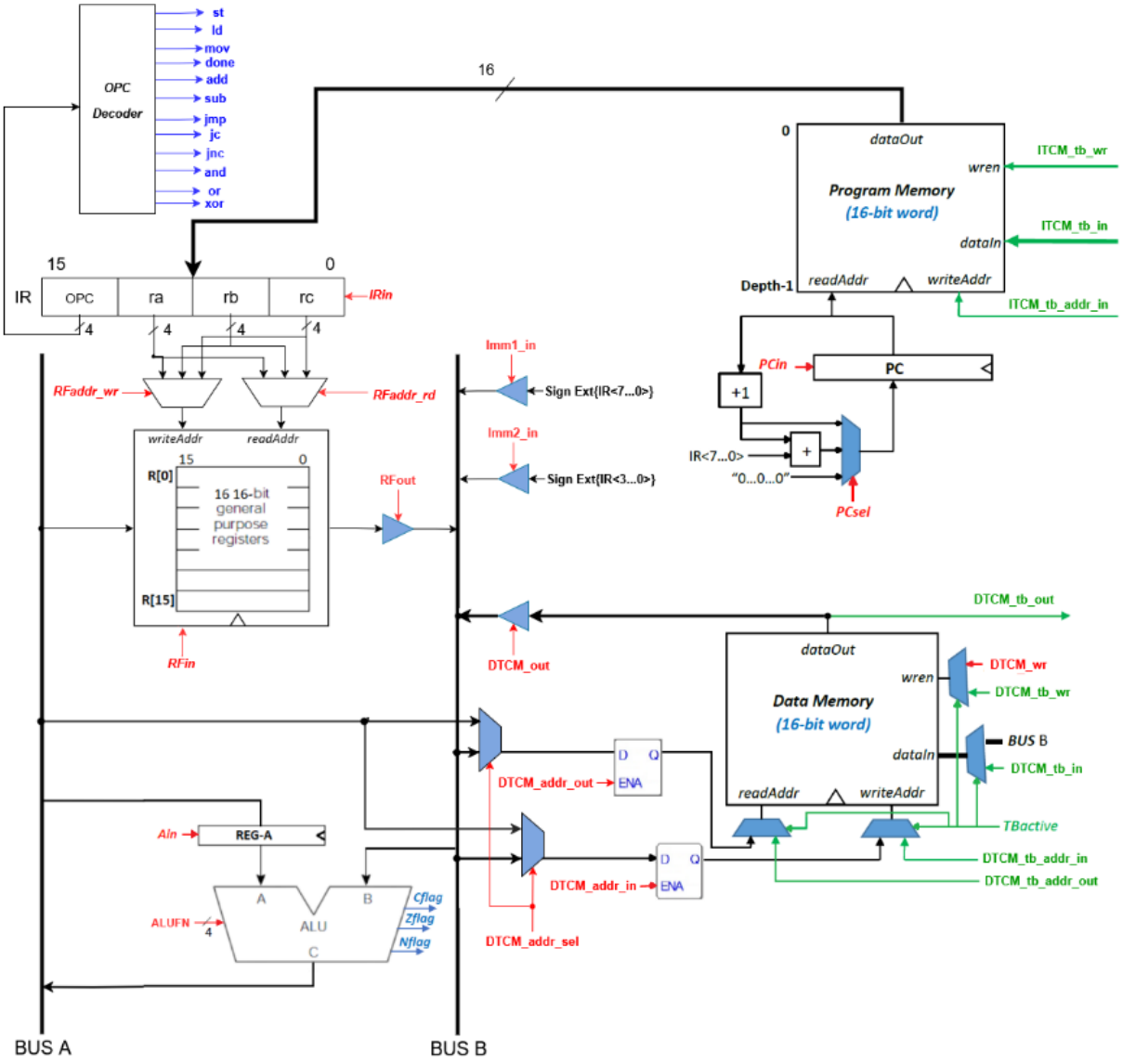
The datapath is the part of the CPU responsible for executing instructions. The instruction is fetched from **Program Memory** into the **Instruction Register (IR)**. The **decoder** interprets the opcode and generates control signals that guide data flow through the registers, ALU, and memories.

This CPU uses **two buses (A and B)**. Values from the **register file** are read onto **bus A**, while **bus B** is used to write results back into the register file or to communicate with memory. The CPU operates in **multiple cycles per instruction**, controlled step by step.

Example Instruction: ld r2, imm(r0)
This loads a word from memory at address R[0] + imm into R[2].

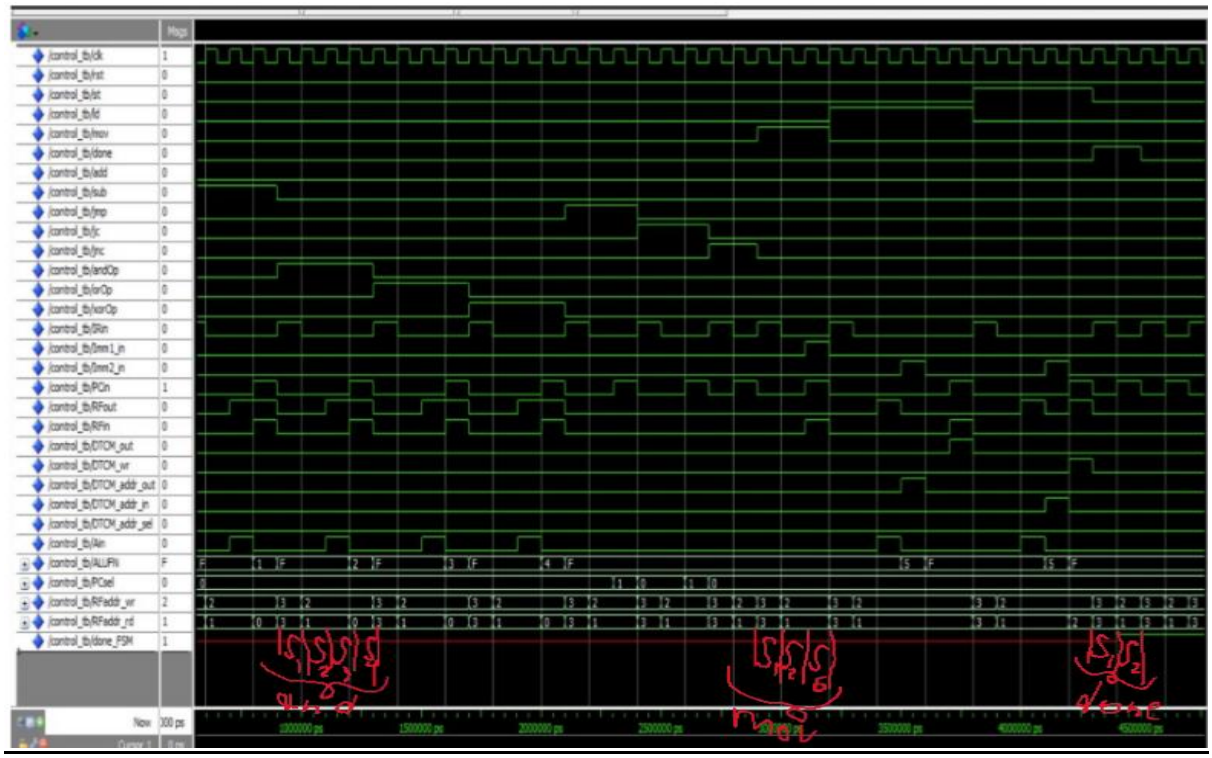
1. **Fetch**:
    - PC provides the address to program memory.
    - Instruction is loaded into IR.
    - PC is incremented.
2. **Decode**:
    - The decoder identifies ld.
    - Register rb = r0 is read from the register file and placed on **bus A**.
3. **Address Calculation**:
    - The immediate value (imm) is sign-extended and placed on **bus B**.
    - The ALU adds R[r0] + imm to form the memory address.
4. **Memory Access**:
    - The address is sent to **Data Memory**, and the word at that location is read.
5. **Write Back**:
    - The memory output is placed on **bus B**.
    - R[r2] is written with this value.

Afterward, control returns to the **Fetch cycle** for the next instruction.


OPC
Decoder
st
ld
mov
done
add
sub
jmp
jc
jnc
and
or
xor
16
0
dataOut
wren
Program Memory
(16-bit word)
dataIn
ITCM_tb_wr
ITCM_tb_in
Depth-1
readAddr
writeAddr
ITCM_tb_addr_in
15
0
IR
OPC
ra
rb
rc
IRin
4
4
4
4
RFaddr_wr
RFaddr_rd
writeAddr
readAddr
15
0
R[0]
16 16-bit
general
purpose
registers
R[15]
RFout
RFin
Imm1_in
Sign Ext{IR<7...0>}
Imm2_in
Sign Ext{IR<3...0>}
PCin
PC
+1
+
IR<7...0>
"0...0...0"
PCsel
DTCM_tb_out
DTCM_out
dataOut
wren
Data Memory
(16-bit word)
dataIn
readAddr
writeAddr
DTCM_wr
DTCM_tb_wr
BUS B
DTCM_tb_in
TBactive
DTCM_tb_addr_in
DTCM_tb_addr_out
D
Q
ENA
DTCM_addr_out
DTCM_addr_in
DTCM_addr_sel
Ain
REG-A
A
B
ALU
C
ALUFN
4
Cflag
Zflag
Nflag
BUS A
BUS B

Simulations

Control Unit

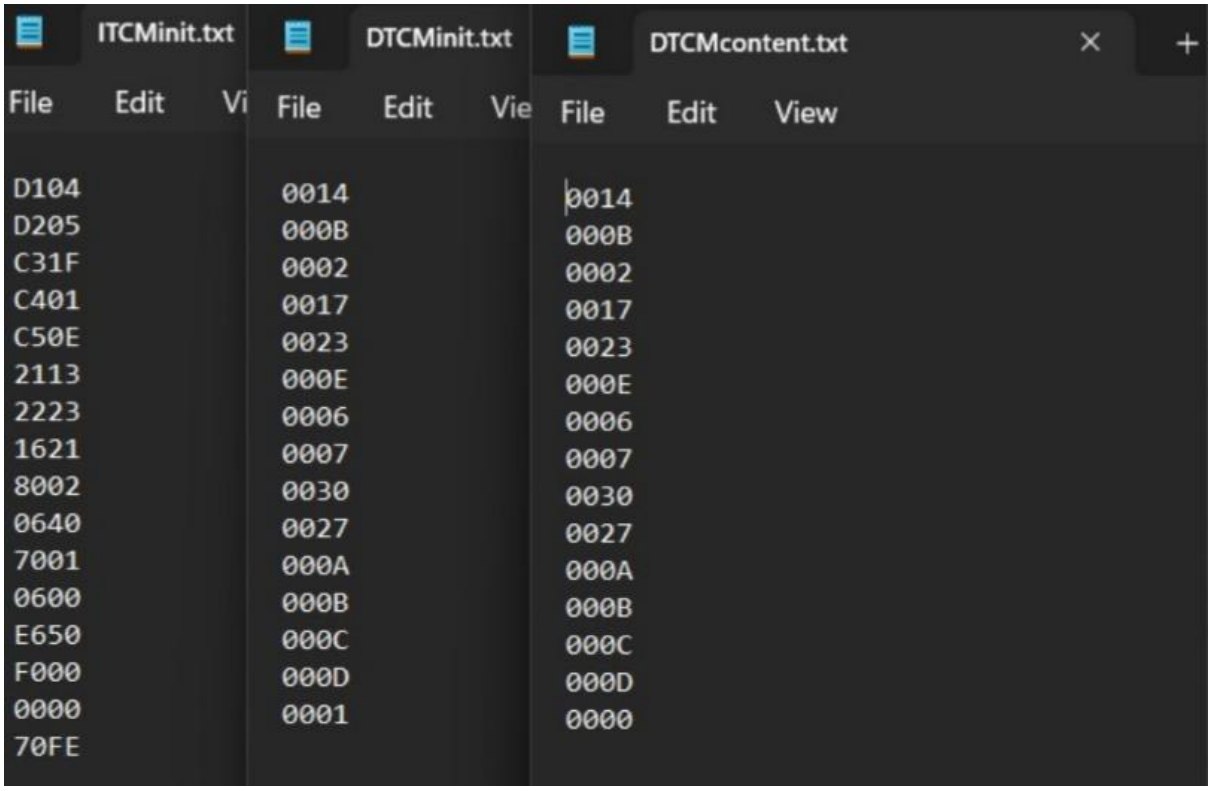


When executing an AND instruction, the Control Unit manages the datapath step by step:

1. **Cycle 1 –** The instruction is loaded from program memory into the IR (IRin).
2. **Cycle 2 –** The opcode is decoded, and the control signals for AND are prepared.
3. **Cycle 3 –** The first source register value is placed on **bus A**.
4. **Cycle 4 –** The second source register value is placed on **bus B**. And the ALU performs the AND operation (OPC = 0010). And then the result from the ALU is returned through **bus B** and written into the destination register.

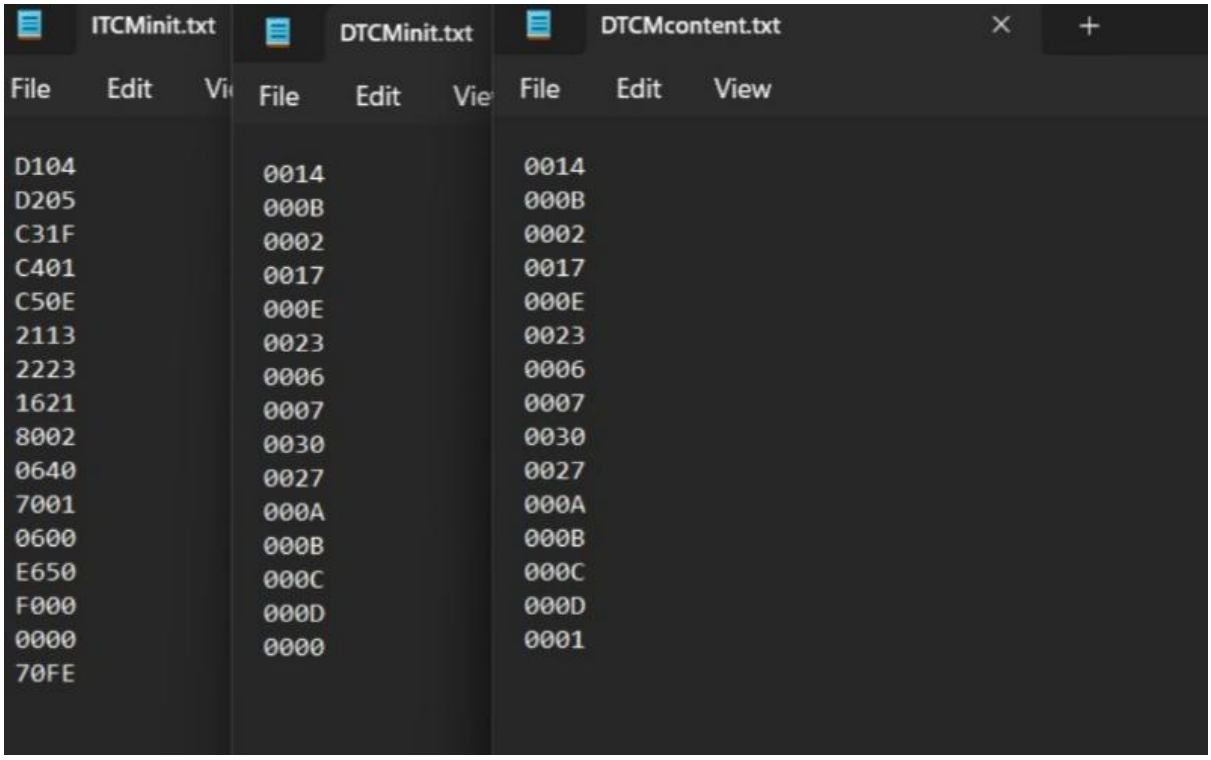
Also mov and done can be seen working like expected which means that the control unit works like expected.

Datapath Unit



It can be seen that it works like expected and stores the expected values in the data memory (the DTCMcontent)

Top (Datapath Unit + Control Unit)



It can be seen that it works like expected and stores the expected values in the data memory (the DTCMcontent)