

DIGITAL DESIGN LAB (EDA322)

CHACC PROCESSOR

Examiner: Prof. Ioannis Sourdis

TAs: Ahsen Ejaz, Ghaith Abou Dan, Henrik Valter, Magnus Östgren, Neethu Bal Mallya, Panagiotis Strikos

Released: Wednesday, January 18, 2023

Contents

1	Introduction	1
2	Instruction Set Architecture	2
3	Datapath	3
3.1	Instruction Execution	3
3.2	Memory	5
3.2.1	Instruction Memory	5
3.2.2	Data Memory	5
3.3	Registers	5
3.4	Arithmetic and Logic Unit (ALU)	6
3.5	Adders	7
3.6	Multiplexers	7
3.7	Bus	7
3.8	7-segment displays	8
4	Controller	9
4.1	Controller's Interface	9
4.2	Control Signals	10
4.3	Examples	11

1 Introduction

This document provides the specifications of the Chalmers Accumulator (ChAcc) processor that will be implemented and evaluated during the seven lab sessions of Digital Design (EDA322). The student will:

1. Implement the processor design in VHDL
2. Verify its implementation using simulation in Modelsim/ Questasim
3. Evaluate the design regarding performance, area, and power dissipation and possibly optimize further
4. Download the processor design on an FPGA

The ChAcc processor, based on the lab processor of HY-120 course in the institute of Computer Science in FORTH, Greece, is a simple and slow processor which can run various programs. It is an 8-bit processor, i.e., the processor executes operations on 8-bit data but executes instructions (machine code in Table 1) which are 12-bit long. ChAcc makes use of the accumulator architecture, which has a special register, called Accumulator (ACC). The register is so named because it can perform consecutive operations (e.g., additions) and accumulate the result. ACC keeps the result of the most recent operation. Almost every instruction works on ACC and the content of a memory location.

This document describes the ChAcc processor and provides important details regarding the Instruction Set Architecture (ISA) and the control signals. This document is organized as follows. Section 2 presents the ISA containing the syntax and the use of the instructions. Section 3 discusses the processor datapath, briefly describing the contained components. Finally, Section 4 discusses the use of the controller documenting the set of control signals and when they must be set/reset so that ChAcc can correctly function.

2 Instruction Set Architecture

The Instruction Set Architecture (ISA) is the set of instructions that a processor can recognize and execute. The ChAcc processor uses its own ISA with 16 instructions as shown in Table 1. The table contains the following columns:

1. Machine code or `imDataOut[11:0]`: The binary code of an instruction which is 12 bits wide wherein the 4 most significant bits compose the opcode (operation code which is unique for each type of instruction) and the 8 least significant bits can be –
 - “aaaaaaa”: 8 bit address (*Addr* in Table) that is used to access the data memory,
 - “coooooo”: 1 bit control and 7 bit offset for the jump instructions like J, JEQ, and JNE, or
 - “xxxxxxx”: or “don’t care” for instructions like NOOP, NOT, and DS as they don’t need to access the data memory or have an offset.
2. Instruction: The name of the instruction
3. Assembly code: The instruction written in assembly language format
4. Comments: A brief description of the instruction and some extra information that must be taken into consideration in particular cases.

Machine Code	Instruction	Assembly Code	Comments
0000xxxxxxx	No Operation	NOOP	Do Nothing
0001xxxxxxx	Input	IN	ACC = Value at IO_BUS
0010xxxxxxx	Display	DS	DS (display register) = ACC
0011coooooo	Jump Equal	JEQ Offset	if (E flag == 1): PC = (PC+1) ± Offset ¹
0100coooooo	Jump Not Equal	JNE Offset	if (E flag == 0): PC = (PC+1) ± Offset ¹
0101coooooo	Jump	J Offset	PC = (PC+1) ± Offset ¹
0110aaaaaaa	Jump Address	JA Addr	PC = Addr
0111aaaaaaa	Compare	CMP ACC, Mem[Addr]	if (ACC == Memory[Addr]): E flag=1 else: E flag=0
1000xxxxxxx	Logical NOT	NOT ACC	ACC = ACC' Set Z flag
1001aaaaaaa	Logical AND	AND ACC, Mem[Addr]	ACC = ACC & Memory[Addr] Set Z flag
1010aaaaaaa	Add	ADD ACC, Mem[Addr]	ACC = ACC + Memory[Addr] Set C and Z flags
1011aaaaaaa	Subtract	SUB ACC, Mem[Addr]	ACC = ACC - Memory[Addr] Set C and Z flags
1100aaaaaaa	Load Byte	LB ACC, Mem[Addr]	ACC = Memory[Addr]
1101aaaaaaa	Store Byte	SB Mem[Addr], ACC	Memory[Addr]=ACC
1110aaaaaaa	Load Byte Index	LBI ACC, Mem[Mem[Addr]]	ACC = Memory[Memory[Addr]]
1111aaaaaaa	Store Byte Index	SBI Mem[Mem[Addr]], ACC	Memory[Memory[Addr]]=ACC

¹ **Offset** is a positive integer contained in `imDataOut[6:0]`, and can be added or subtracted to `pcOut` (which is already updated to (PC+1) in the FE stage, see Figure 1) based on the *c* bit in the instruction, i.e. `imDataOut[7]`. 0 means addition and 1 means subtraction.

Table 1: Instruction Set Architecture (ISA) of the ChAcc processor

ChAcc processor primarily consists of three groups of instructions:

1. **Arithmetic and logic instructions:** The instructions Add, Subtract, AND, NOT and Compare belong to this group. These instructions make use of the ALU unit and perform arithmetic or logic operations between the ACC and the content of a data memory location (except the NOT instruction). The address field of the instruction is used to access the data memory and retrieve the second data operand for the ALU.

2. **Memory instructions:** The instructions Load Byte, Store Byte, Load Byte Index and Store Byte Index that belong to this group access the data memory using the address field of the instruction as an index. Memory instructions can:
 - (a) read something from the data memory and save it to the ACC (Load Byte, Load Byte Index),
 - (b) write the content of the ACC into the data memory (Store Byte, Store Byte Index), or
 – Note that the instructions LBI and SBI access the data memory twice.
3. **Jump instructions:** The instructions Jump, Jump Address, Jump Equal and Jump Not Equal that belong to this group can change the program flow by modifying the program counter (PC) based on a condition (JEQ, JNE) or unconditionally (J), by jumping to a particular address (JA). The address of the next instruction is calculated based on the instruction.
4. **Misc instructions:** There are three other instructions that do not belong to any of the groups above.
 - DS is used for debugging by copying the content of the ACC register into the Display (DS) register)
 - IN is used to write the data that come from the I/O bus into the ACC register
 - NOOP is used to keep the processor idle. It does nothing

3 Datapath

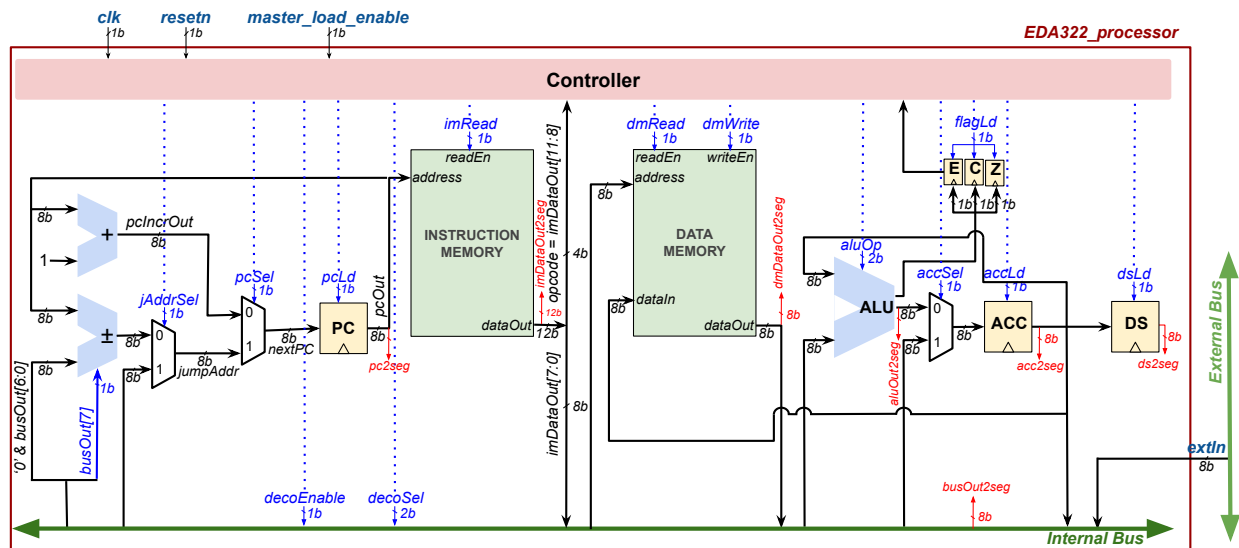


Figure 1: ChAcc processor datapath

The datapath is the portion of the processor that contains hardware components necessary to execute instructions by the processor. The ChAcc datapath is depicted in Figure 1. The datapath consists of many different components such as memory, registers, an Arithmetic and Logic Unit (ALU), adders, multiplexers, buses, controller and the 7-segment displays. The controller (Section 4) is the brain of the processor since it orchestrates the different units based on the executed instruction (Section 2).

3.1 Instruction Execution

The ChAcc processor, like any processor, runs a set of instructions or in other words a program. The program is executed instruction-by-instruction. An instruction execution can be split into the following stages:

- **(FE) Instruction Fetch:**
 - The instruction (`imDataOut`) is read from the Instruction Memory using program counter (PC) as the address
 - PC is incremented to the address of next sequential instruction

- **(DE1) Instruction Decode and Data Fetch:**
 - The controller decodes the opcode (`imDataOut[11:8]`) to figure out which processor's units will be used and which control signals must be set or unset during the whole instruction's execution
 - The data is fetched from the Data Memory using the address part of the instruction (`imDataOut[7:0]`)
- **(DE2) Second Data Fetch:**
 - The Data Memory is read for a second time to get the data for load index instruction (LBI)
- **(EX) Execute:**
 - Arithmetic and logic instructions (e.g., ADD, SUB, NOT, AND, CMP) are executed using Arithmetic-Logic Unit (ALU), and the result is saved into the ACC register (except for CMP which updates only the E register).
 - Load instructions (e.g., LB, LBI) update the value of ACC register with the data read from memory
 - Input instruction (IN) writes the value from external input to the ACC register
 - DS instruction sets the `dsLd` control signal and completes execution
 - Jump instructions (e.g., JA, J, JEQ, JNE) calculates the jump address (`jumpAddr`), updates PC and completes execution
- **(ME) Write to Memory:**
 - Store instructions (e.g., SB, SBI) writes the previously calculated result (already saved in ACC) to the memory

Each instruction uses different datapath components during execution and thus, may not require all five stages. Table 2 summarizes the stages utilized by the different instructions marking with ✓ for the used ones and with ✗ for the unused stages. Every stage has a duration of one clock cycle. The last column of the table presents the actual number of used stages (cycles needed) per instruction.

Opcode	Assembly Code	FE	DE1	DE2	EX	ME	#stages
0000	NOOP	✓	✓	✗	✗	✗	2
0001	IN	✓	✓	✗	✓	✗	3
0010	DS	✓	✓	✗	✓	✗	3
0011	JEQ Offset	✓	✓	✗	✓	✗	3
0100	JNE Offset	✓	✓	✗	✓	✗	3
0101	J Offset	✓	✓	✗	✓	✗	3
0110	JA Addr	✓	✓	✗	✓	✗	3
0111	CMP ACC, Mem[Addr]	✓	✓	✗	✓	✗	3
1000	NOT ACC	✓	✓	✗	✓	✗	3
1001	AND ACC, Mem[Addr]	✓	✓	✗	✓	✗	3
1010	ADD ACC, Mem[Addr]	✓	✓	✗	✓	✗	3
1011	SUB ACC, Mem[Addr]	✓	✓	✗	✓	✗	3
1100	LB ACC, Mem[Addr]	✓	✓	✗	✓	✗	3
1101	SB Mem[Addr], ACC	✓	✓	✗	✗	✓	3
1110	LBI ACC, Mem[Mem[Addr]]	✓	✓	✓	✓	✗	4
1111	SBI Mem[Mem[Addr]], ACC	✓	✓	✗	✗	✓	3

Table 2: Datapath stages per instruction

The clock cycle time of the processor is determined by the latency of the slowest datapath stage (critical path). If the whole datapath was clocked as one large stage, all the instructions would have the same execution time resulting in a simpler controller design. However, it is more advantageous to have a multi-stage datapath as different instructions of the ISA utilize a variable number of datapath stages, thus requiring a variable number of clock cycles, resulting in different execution times among them. This can potentially yield a more efficient design in terms of performance. Finally, a multi-stage datapath can be more easily pipelined to parallelize the execution of more instructions per cycle. However, the latter requires computer organization knowledge and is out of the scope of this course. The rest of this section focuses on particular components of the datapath.

3.2 Memory

The ChAcc datapath contains two memories, Instruction Memory and Data Memory, which store the program's instructions and data, respectively. Each memory is accessed using an 8-bit address, implying that each has 2^8 entries (= 256 entries). Both memories are synchronous, meaning that all accesses must be clock synchronized. Table 3 and 4 describe the inputs and outputs of the Instruction Memory and Data Memory.

3.2.1 Instruction Memory

- Each entry in the Instruction Memory is 12 bits
- The memory takes an input address (*address*)
- If *readEn* is enabled at a rising clock edge, then data is read from *address* to the *dataOut* port
- The memory can be initialized using a memory initialization file (*-mif*) in the implementation

Name	#bits	Type	Comments
clk	1	Input	The processor's clock signal
readEn	1	Input	Reads data from <i>address</i> to <i>dataOut</i> port, when set
address	8	Input	The memory address for memory read operation
dataOut	12	Output	The output data from memory for a memory read operation

Table 3: Inputs and outputs of the Instruction Memory

3.2.2 Data Memory

- Each entry in the Data Memory is 8 bits
- The memory takes an input address (*address*)
- If *writeEn* is enabled at a rising clock edge, then data is written from the *dataIn* port to *address*
- If *readEn* is enabled at a rising clock edge, then data is read from *address* to the *dataOut* port
- The memory can be initialized using a memory initialization file (*-mif*) in the implementation

Name	#bits	Type	Comments
clk	1	Input	The processor's clock signal
writeEn	1	Input	Writes data from <i>dataIn</i> port to <i>address</i> , when set
readEn	1	Input	Reads data from <i>address</i> to <i>dataOut</i> port, when set
address	8	Input	The memory address for memory read/write operation
dataIn	8	Input	The input data to memory for a memory write operation
dataOut	8	Output	The output data from memory for a memory read operation

Table 4: Inputs and outputs of the Data Memory

3.3 Registers

The register is the simplest storage component used in the ChAcc processor. It contains a D flip-flop which stores the input in each positive clock edge. All the registers receive an input control signal (*loadEnable*) from the controller. Based on the control signal, the register either maintains the current value or updates it with a new value. Figure 2 shows a 1-bit register using D flip-flop. An *n-bit* register can store an *n-bit* value, thus having *n* number of flip-flops. Table 5 describe the inputs and outputs of an *n-bit* register.

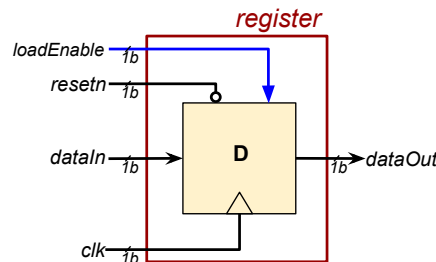


Figure 2: 1-bit register using D flip-flop

Name	#bits	Type	Comments
dataIn	n	Input	Data input of the register
clk	1	Input	Connected to the processor's clock
resetn	1	Input	Connected to the processor's reset signal which is asynchronous and active when it is 0.
loadEnable	1	Input	Control input signal which updates the register with a new input, when set (i.e., $\text{loadEnable} = 1$)
dataOut	n	Output	Data output of the register

Table 5: Inputs and outputs of an n -bit Register

The registers of the ChAcc processor are described in Table 6. As shown in Figure 1, all the registers receive an input control signal (or loadEnable) from the controller.

Name	#bits	Control Signal	Comments
PC	8	pcLd	Stores the address of next instruction in the program sequence
ACC	8	accLd	Stores the value of the recent ALU operation
DS	8	dsLd	Stores the content of the ACC if we decide to show its value on the FPGA's display, using the instruction DS
E	1	flagLd	Stores the flag which indicates that the two ALU inputs are equal
C	1	flagLd	Stores the flag which indicates the carry in the ALU operation
Z	1	flagLd	Stores the flag which indicates that the ALU output is zero

Table 6: List of registers in the ChAcc processor

3.4 Arithmetic and Logic Unit (ALU)

The datapath contains an Arithmetic and Logic Unit (ALU) to perform all the necessary arithmetic and logic operations. In most modern processors, the ALU can perform arithmetic operations such as addition, subtraction, multiplication, and division on integer and floating-point operands and logic operations. However, the ALU of the ChAcc processor is relatively simple and only performs addition, subtraction, and a few logic operations (AND, NOT, and compare). Furthermore, our ALU supports arithmetic operations only between unsigned numbers.

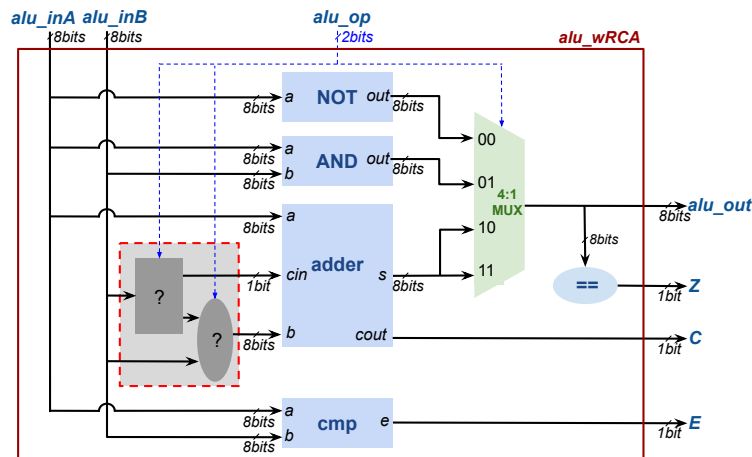


Figure 3: Block Diagram of the ALU

The block diagram of the ALU is depicted in Figure 3. The ALU has three inputs and four outputs as described in Table 7. The sub-components perform operations as listed in Table 8 based on alu_op . Note that **cmp** compares alu_inA and alu_inB and is always active irrespective of the operation.

Name	#bits	Type	Comments
alu_inA	8	Input	First data operand
alu_inB	8	Input	Second data operand
alu_op	2	Input	Control signal that determines the ALU operation
alu_out	8	Output	Output of the ALU operation
C flag	1	Output	Carry out <i>cout</i> of the adder
E flag	1	Output	Output of the compare operation; Set when the input data operands are equal
Z flag	1	Output	Set when the ALU output is zero.

Table 7: Inputs and outputs of the ALU

alu_op	Operation	Action
00	Logical NOT	$\text{alu_out} = \text{alu_inA}'$
01	Logical AND	$\text{alu_out} = \text{alu_inA} \& \text{alu_inB}$
10	Addition	$\text{alu_out} = \text{alu_inA} + \text{alu_inB}$
11	Subtraction	$\text{alu_out} = \text{alu_inA} - \text{alu_inB}$

Table 8: ALU Operations for given ALU_op

3.5 Adders

The datapath includes two adders, which does the following operations:

(Adder 1) Increments the PC to compute the address of the next sequential instruction

$$\text{pcIncrOut} = \text{pcOut} + 1$$

(Adder 2) Computes the jump target address for J, JEQ and JNE instructions

$$\text{jumpAddr} = \text{pcOut} \pm \text{Offset}$$

Offset is a positive integer contained in `imDataOut[6:0]`, and can be added or subtracted to `pcOut`.

- If *c* bit in the instruction, i.e. `imDataOut[7]` is 0, $\text{jumpAddr} = \text{pcOut} + \text{Offset}$
- If *c* bit in the instruction, i.e. `imDataOut[7]` is 1, $\text{jumpAddr} = \text{pcOut} - \text{Offset}$

3.6 Multiplexers

The datapath includes three 2:1 multiplexers (*mux*) to select between two inputs based on the control signal generated by the controller. Table 9 describes the different muxes in the ChAcc datapath with the two inputs and control signals.

Name	Description	Select Signal	Inputs
jump Address mux	Selects jump address	jAddrSel	Input0: Jump address based on offset for J, JEQ and JNE, which is the output of the adder Input1: Jump address for JA (<code>imDataOut[7:0]</code>)
pc mux	Selects address for the next instruction	pcSel	Input0: <code>pcIncrOut</code> Input1: <code>jumpAddr</code>
ACC mux	Selects the source of ACC register	accSel	Input0: ALU output Input1: Output of the bus (<code>busOut</code>) for load and IN instructions

Table 9: Multiplexers in the ChAcc datapath

3.7 Bus

On the bottom of Figure 1, we can see the internal bus of the ChAcc processor. The bus communicates data between different components in the datapath. The bus has 6 inputs (4 data inputs, 1 enable input, 1 control input) and 1 data output as discussed in Table 10.

Name	#bits	Type	Comments
imDataOut	8	Data Input	Source: 8 lower significant bits of the Instruction Memory output, i.e., imDataOut[7:0]
dmDataOut	8	Data Input	Source: Output from the Data Memory, dmDataOut
accOut	8	Data Input	Source: Output from the accOut register
extIn	8	Data Input	Source: Output from the external bus, extIn
decoEnable	1	Enable Input	Enable input of the decoder from the controller
decoSel	2	Control Input	Decoder control signal from the controller
busOut	8	Data Output	The bus data output

Table 10: Inputs and output of the Bus

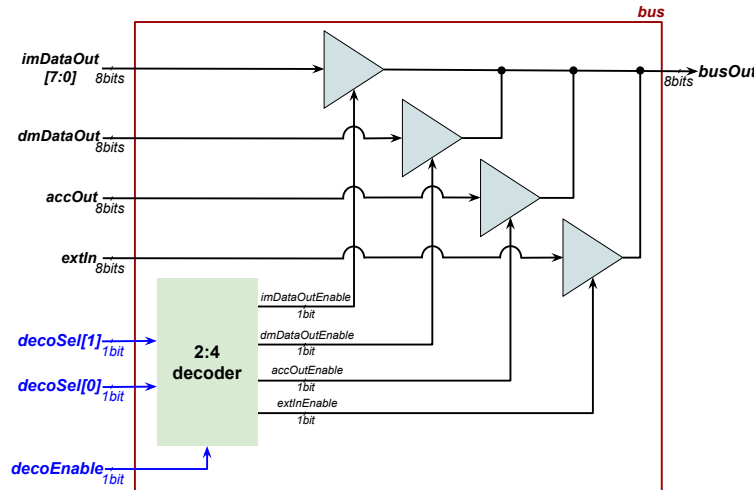


Figure 4: Bus with decoder and tri-state buffers

The bus is implemented with a 2:4 decoder and four tri-state buffers as shown in Figure 4. The four different data inputs are connected to the tri-state buffers and the outputs generated by the four buffers are connected to form a single bus line. The control inputs to the buffers determine which of the four data inputs will communicate with the bus line. Note that only one buffer can be in active state at a given point in time. This is ensured by the 2:4 decoder which generates the control inputs to the buffers and hence, no more than one control input is active at a given point in time. The truth table of the decoder is presented in Table 11. When **decoEnable** is 0, all of the four outputs are 0, and the bus line is in a high-impedance state because all four buffers are disabled. When **decoEnable** is active, one of the three-state buffers will be active, depending on the binary value in the select inputs of the decoder.

decoEnable	decoSel[1]	decoSel[0]	imDataOutEnable	dmDataOutEnable	accOutEnable	extInEnable
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Table 11: Truth table of the 2:4 decoder

3.8 7-segment displays

Many datapath signals are connected to 7-segment displays (self-explanatory names), as depicted in Figure 1. The user can use these displays to track the value of particular signals or registers to verify the correct operation when the processor is running. The 7-segment displays are handy when debugging the design.

4 Controller

In any processor, a special unit is needed in order to synchronize the rest of the components in the datapath and orchestrate their operations. This unit is called controller and is actually the “brain” of a processor. In the ChAcc processor, the controller is shown on the top of Figure 1. The controller is implemented as a Finite State Machine (FSM), and the details are provided in the Lab 4 assignment. This section discusses how the controller controls the components during the different stages of an instruction execution.

4.1 Controller’s Interface

As different instructions make use of different datapath stages, the controller must determine which datapath stage is used by an instruction and when (which cycle), by setting/resetting particular signals (marked in blue color in Figure 1) that control the various datapath components like the muxes, the bus, the registers, the memory, the ALU, etc. The inputs and outputs are detailed in Table 12 and Table 13, respectively.

Name	#bits	Comments
clk	1	Synchronizes every sequential circuit (the processor’s clock)
resetn	1	Initializes the components when active ($\text{RESETn} = 0$). Reset is <i>asynchronous</i> , i.e., not dependent on the rising/falling edge of the clock.
master_load_enable	1	This signal is connected to an FPGA switch and can be used to control manual clock toggling. In other words, by toggling this signal, the user is able to control the clocking of the design, <i>freezing</i> and <i>starting</i> the time. This is useful when debugging the design; otherwise the changes on the displays would not be visible to a human’s eye, as the design’s clock is on the order of hundreds of MHz. The master_load_enable affects the following: <ol style="list-style-type: none"> 1. The internal state transitions of the controller (FSM) are enabled when master_load_enable is set. 2. The registers save their input on the rising clock edge only when <ul style="list-style-type: none"> • master_load_enable is set (master_load_enable = 1) • respective control signal of a register is set • RESETn is disabled (i.e., RESETn = 1)
opcode	4	The four most significant bits of the current instruction (imDataOut [11:8]) to set/reset the control signals during different stages of the instruction execution.
E flag	1	Output from E register; Used for conditional jump instructions

Table 12: ChAcc Controller Inputs

Name	#bits	Component	Comments
decoEnable	1	Bus	Enables decoder inside the internal bus, when set
decoSel	2	Bus	Controls the decoder inside the internal bus
jAddrSel	1	jump Address mux	Select signal for jump Address mux
pcSel	1	pc mux	Select signal for pc mux
accSel	1	ACC mux	Select signal for ACC mux
aluOp	2	ALU	Control signal that determines the ALU operation
imRead	1	Memory	Enables the read function of the Instruction Memory, when set
dmRead	1	Memory	Enables the read function of the Data Memory, when set
dmWrite	1	Memory	Enables the write function of the Data Memory, when set
pcLd	1	PC register	Enables PC register, when set
flagLd	1	E, C and Z registers	Enables E, C and Z registers, when set
accLd	1	ACC register	Enables ACC register, when set
dsLd	1	DS register	Enables the load of the DS register

Table 13: ChAcc Controller Outputs

4.2 Control Signals

Figure 5 depicts the values of the control signals for every instruction. The first column of the table presents the *opcode* (of the decoded instruction) while the row summarizes all the control signals for the corresponding *opcode*. Use the notation presented below to understand the control signals in Figure 5.

- The signal is presented in **X_Y** format, where **X** is the signal's value and **Y** is the stage at which the signal must take this value.
- Only the value **X** is presented when the signal is either set or unset during the whole execution of the instruction. *Example:* For *opcode* = 1000, *decoEnable* is 0 during the whole execution of instruction.
- The value of a signal may be *x* (don't care) instead of 1 or 0 which means that it can take any value. *Example:* For *opcode* = 1010, *jAddrSel* is never used and thus is marked as **x**.
- For **JEQ** and **JNE** instructions: The control signals in the **EX** stage are updated only if the respective flag condition is met (highlighted in **Only if ..**).
- All the control signals should have a default value (say, set to 0 at boot-up)

Opcode	Instruction	decoEnable	decoSel	jAddrSel	pcSel	accSel	aluOp	imRead	dmRead	dmWrite	pcLd	flagLd	accLd	dsLd
Active only if master_load_enable = 1														
0000	NOOP	0	xx	x	0_FE	x	xx	1_FE	0	0	1_FE	0	0	0
0001	IN	1_EX	11_EX	x	0_FE	1_EX	xx	1_FE	0	0	1_FE	0	1_EX	0
0010	DS	0	xx	x	0_FE	x	xx	1_FE	0	0	1_FE	0	0	1_EX
0011	JEQ <small>OnlyIf E=1</small>	1_EX	00_EX	0_EX	0_FE 1_EX	x	xx	1_FE	0	0	1_FE 1_EX	0	0	0
0100	JNE <small>OnlyIf E=0</small>	1_EX	00_EX	0_EX	0_FE 1_EX	x	xx	1_FE	0	0	1_FE 1_EX	0	0	0
0101	J	1_EX	00_EX	0_EX	0_FE 1_EX	x	xx	1_FE	0	0	1_FE 1_EX	0	0	0
0110	JA	1_EX	00_EX	1_EX	0_FE 1_EX	x	xx	1_FE	0	0	1_FE 1_EX	0	0	0
0111	CMP	1_DE1 1_EX	00_DE1 01_EX	x	0_FE	x	xx	1_FE	1_DE1	0	1_FE	1_EX	0	0
1000	NOT	0	xx	x	0_FE	0_EX	00_EX	1_FE	0	0	1_FE	1_EX	1_EX	0
1001	AND	1_DE1 1_EX	00_DE1 01_EX	x	0_FE	0_EX	01_EX	1_FE	1_DE1	0	1_FE	1_EX	1_EX	0
1010	ADD	1_DE1 1_EX	00_DE1 01_EX	x	0_FE	0_EX	10_EX	1_FE	1_DE1	0	1_FE	1_EX	1_EX	0
1011	SUB	1_DE1 1_EX	00_DE1 01_EX	x	0_FE	0_EX	11_EX	1_FE	1_DE1	0	1_FE	1_EX	1_EX	0
1100	LB	1_DE1 1_EX	00_DE1 01_EX	x	0_FE	1_EX	xx	1_FE	1_DE1	0	1_FE	0	1_EX	0
1101	SB	1_ME	00_ME	x	0_FE	x	xx	1_FE	0	1_ME	1_FE	0	0	0
1110	LBI	1_DE1 1_DE2 1_EX	00_DE1 01_DE2 01_EX	x	0_FE	1_EX	xx	1_FE	1_DE1 1_DE2	0	1_FE	0	1_EX	0
1111	SBI	1_DE1 1_ME	00_DE1 01_ME	x	0_FE	x	xx	1_FE	1_DE1	1_ME	1_FE	0	0	0

Figure 5: ChAcc Control Signals

4.3 Examples

Let's take some example instructions to explain how particular control signals are set or unset. You can better comprehend these examples by looking at the datapath animations in the slides of Lecture 2 and the following paragraphs.

Example 1) ADD instruction (opcode: 1010): As given in Table 2, the ADD instruction uses 3 stages – **FE**, **DE1** and **EX** as follows:

- **FE:** A read from the instruction memory is issued using program counter (**pcOut**) as the Instruction Memory address (**imAddress**) and PC is incremented to the address of next sequential instruction.
- **DE1:** The controller decodes the opcode (**imDataOut[11:8]**) to figure out which processor's units will be used and which control signals must be set or unset during the whole instruction's execution. The data (or input operand) is fetched using the address part of the instruction from the memory, i.e., **dmAddress = imDataOut[7:0]**.
- **EX:** The ALU performs the addition operation (**aluOp = 10**) on the output of ACC register (**accOut**) and the output of the bus (**busOut**). For addition, the **busOut** will be the value from **dmDataOut** (holding the value of the address set in the **DE1** stage). So we set the decoder inputs of the bus accordingly. The output of the ALU operation is updated to **accOut**.

The control signals for the operations per stage are given below:

Stage	Operation	Control Signal(s)
FE	imAddress = pcOut	imRead = 1
	pcIncrOut = pcOut + 1	
	nextPC = pcIncrOut	pcSel = 0
	PC = nextPC	pcLd = 1
DE1	Controller decodes imDataOut[11:8]	
	dmAddress = imDataOut[7:0]	decoEnable = 1, decoSel = 00, dmRead = 1
EX	aluOut = accOut + busOut	decoEnable = 1, decoSel = 01
	where busOut = dmDataOut	aluOp = 10
	ACC = aluOut	accSel = 0, accLd = 1
	Set C and Z flags	flagLd = 1

Table 14: Control Signals for ADD

Example 2) LBI instruction (opcode: 1110): As given in Table 2, the LBI instruction uses 4 stages – **FE**, **DE1**, **DE2** and **EX** as follows:

- **FE:** A read from the instruction memory is issued using program counter (**pcOut**) as the Instruction Memory address (**imAddress**) and PC is incremented to the address of next sequential instruction.
- **DE1:** The controller decodes the opcode (**imDataOut[11:8]**) to figure out which processor's units will be used and which control signals must be set or unset during the whole instruction's execution. The data (or input operand) is fetched using the address part of the instruction from the memory, i.e., **dmAddress = imDataOut[7:0]**.
- **DE2:** The data for the load index operation is fetched from the memory using the address from **dmDataOut** i.e., **dmAddress = dmDataOut**, and **dmDataOut** is updated with the value from Data memory
- **EX:** The value of ACC register (**accOut**) is updated with the output of the bus (**busOut**). For load, the **busOut** will be the value from **dmDataOut** made available from **DE2** stage. So we set the decoder inputs of the bus accordingly.

The control signals for the operations per stage are given below:

Stage	Operation	Control Signal(s)
FE	$\text{imAddress} = \text{pcOut}$	$\text{imRead} = 1$
	$\text{pcIncrOut} = \text{pcOut} + 1$	
	$\text{nextPC} = \text{pcIncrOut}$	$\text{pcSel} = 0$
	$\text{PC} = \text{nextPC}$	$\text{pcLd} = 1$
DE1	Controller decodes $\text{imDataOut}[11:8]$	
	$\text{dmAddress} = \text{imDataOut}[7:0]$	$\text{decoEnable} = 1, \text{decoSel} = 00, \text{dmRead} = 1$
DE2	$\text{dmAddress} = \text{dmDataOut}$	$\text{decoEnable} = 1, \text{decoSel} = 01, \text{dmRead} = 1$
EX	$\text{ACC} = \text{busOut}$	$\text{decoEnable} = 1, \text{decoSel} = 01$
	where $\text{busOut} = \text{dmDataOut}$	$\text{accSel} = 1, \text{accLd} = 1$

Table 15: Control Signals for LBI

Example 3) J instruction (opcode: 0101): As given in Table 2, the LBI instruction uses 4 stages – **FE**, **DE** and **EX** as follows:

- **FE:** A read from the instruction memory is issued using program counter (pcOut) as the Instruction Memory address (imAddress) and PC is incremented to the address of next sequential instruction.
- **DE:** The controller decodes the opcode ($\text{imDataOut}[11:8]$) to figure out which processor's units will be used and which control signals must be set or unset during the whole instruction's execution.
- **EX:** The jump Address is calculated using the adder ($\text{jumpAddr} = \text{pcOut} \pm \text{Offset}$) and PC is updated ($\text{nextPC} = \text{jumpAddr}$ and PC register = nextPC). The control signal to the adder is $\text{busOut}[7]$ i.e., $\text{imDataOut}[7]$ of the jump instruction. So we set the decoder inputs of the bus accordingly.

The control signals for the operations per stage are given below:

Stage	Operation	Control Signal(s)
FE	$\text{imAddress} = \text{pcOut}$	$\text{imRead} = 1$
	$\text{pcIncrOut} = \text{pcOut} + 1$	
	$\text{nextPC} = \text{pcIncrOut}$	$\text{pcSel} = 0$
	$\text{PC} = \text{nextPC}$	$\text{pcLd} = 1$
DE	Controller decodes $\text{imDataOut}[11:8]$	
EX	$\text{jumpAddr} = \text{pcOut} \pm \text{busOut}$	$\text{decoEnable} = 1, \text{decoSel} = 00$
	where $\text{busOut} = \text{imDataOut}[7:0]$	
	$\text{jumpAddr} = \text{Output of the adder}$	$\text{jAddrSel} = 0$
	$\text{nextPC} = \text{jumpAddr}$	$\text{pcSel} = 1$
	$\text{PC} = \text{nextPC}$	$\text{pcLd} = 1$

Table 16: Control Signals for J

Finally, it must be mentioned here that the purpose of this document was to provide you with the specifications of the ChAcc processor, which included the overview of the processor's datapath, ISA, and components. We will provide the functionality and the interfaces of particular components (e.g., adder, controller FSM) in the corresponding lab assignments.

