Workshop 1: 4-bit CPU Design

Charles Khoury
charles.khoury@bristol.ac.uk

January 9, 2023

1 Introduction

The goal of this workshop is to gain a better understanding of the inner-workings of the 4-bit CPU and understand the design philosophy. The CPU is designed using ModuleSim which simulates hardware previously used in the delivery of the introductory unit to computer architecture for undergraduates, the modules were meant to work in tandem with the simulation; hence the unusual component choice. This implies the need to work around the design constrains set by the design tool.

2 Basic design knowledge

Fundamentally CPUs are Finite State Machines (FSM) containing the following component block:

- Control Unit (CU).
- Arithmetic Logic Unit (ALU).
- Registers.
- Buses (internal data lines).
- Clock
- Cache

The processor in this unit doesn't have a cache, instead instructions are fetched directly from memory (NRAM component in ModuleSim).
ALU: performs arithmetic and logical operation and in for the 4-bit processor the part of the multistage memory write process (see STAM).
CU: Performs the Fetch, Decode, Execute operations by issuing control signals to the hardware and moving data around.
Registers: Small amount of memory contained within the CPU to store data needed to perform the various functions. Such as instruction address, current opcode and optional operand, result of calculations. Modern processors possess specific function registers (such as the Program Counter (PC), Instruction Register, etc.) and general purpose register than can be used to store addresses, computation results etc.
Clock: Used to synchronize the different CPU components.
This computational model is what is know as the Von Neumann architecture.

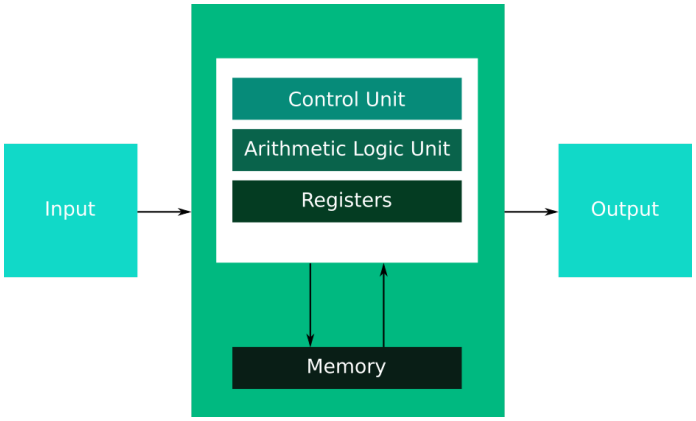

Figure 1: Von Neumann Architecture diagram

3 Understanding the Instruction Set

3.1 Instruction Set Architecture

The base implementation of the 4-bit uses 8 instructions encoded over 4 bits.
Question: Theoretically, how many instructions can be encoded over 4 bits?

Mnemonic	Operands/Example	Description
`LDAC`	*Cn*	Load constant in register A
	`LDAC 5`	Load 5 into register A
`LDBC`	*Cn*	Load constant in register B
	`LDBC 5`	Load 5 into register B
`LDAM`	*Addr*	Load LSB at address of *Addr* into register A
	`LDAM 10`	Load LSBs stored in memory location 10 into register A
`LDBM`	*Addr*	Load LSBs at address of *Addr* into register B
	`LDBM 10`	Load LSBs stored in memory location 10 into register B
`STAM`	*Addr*	Store the content register A in memory location *Addr* and reset the MSB
	`STAM 10`	Store content of A
`ADD`		Add the values stored in register A and B and store the result in A
	`ADD` *A = 6, B=4,*	Value of A after Phase 3b : 10
`SUB`		Substract the values stored in register A and B and store the result in A
	`SUB` *A = 6, B=4,*	Value of A after Phase 3b : 2
`LDAI`	*offset*	Load the content from memory at address [A+offset] into A
	`LDAI` LDAI 4 *A=6,Mem10 = 12*	Load 12 into register A

Table 1: 4-bit CPU ISA

3.2 Exercise

Question: What is the value stored in register B after complete execution of the following code:

```
LDAC 2
LDBC 3
ADD
STAM 10
LDBC 4
SUB
LDBM 10
```

Answer: The value in register B since the result of A+B (A=2, B=3) is stored in address 10 which is then loaded in register B in the final instruction
Question: Consider the code below:

```
LDAC 4
LDBC 3
STAM 3
ADD
```

What is the value of register A after complete execution of this code.
Answer: The final value is 4 and NOT 7 as the STAM instruction overwrites the content at location 0x03 which is where the ADD instruction is located in this code.
Note: Try to understand how writing to memory affected code execution.
Question: Which instruction(s) write to Register A?
Answer: LDAC, LDAI, LDAM, ADD, SUB
Question: Which instruction(s) read and write to/from the main memory ?
Answer: LDBM, LDAI, LDAM, STAM,

4 Design Implementation

4.1 Arithmetic Logic Unit

4.2 Control Unit

As mentioned above the role of the CU is to perform the the Fetch, Decode, Execute operation by deriving control signals.
Question: What are the different entit(y/ies) that need to be controlled in the data path for each instruction ?
Derive the control signal needed for each instructions to the different components of the data path. Include a fetch and decode phase as part of your work.
Answer: Please refer to the file $ISA_control.pdf$
$For this processor the CU is implemented in two part, a phase generator and decoder unit. The former allows to select$
$Execute]) and the latter allows you generate a control signal based on the instruction (i.e. Decode).$

4.2.1 Clock Phases

The segmentation of the Fetch, Increment PC and [Decode-Execute] is done using the two phase clock and a counter to generate an upper phase. The main clock has 2 non-overlapping phases (a, b).

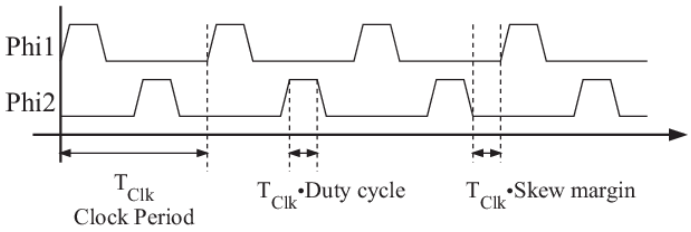

Figure 2: Non overlapping clock

In ModuleSim, when connected to a register the clock module is able to generate the control signal to the register as to write or reset it. From there, a counter tied to phase b is used to generate a numbered phased $(1, 2, 3)$ *(The counter can generate more phases which remain unused).*
Question: *By combining numbered and lettered phases, how many phases can be generated ?*

4.2.2 Instruction Decoding

The decoding process determines what instruction is to be performed and produces a sequence on each control line top implement functionality. This is typically done using a binary encoder that "1-hot" encodes the signal. For the processor studied in this unit this is achieved using multiple demultiplexers.
Question: *Why do we need multiple demultiplexers ?*
Answer: *The DMX module is a 2-4 demux with a 2 bit control signal (each lane represents a 4 bit bus). Since we need to implement more than 4 instructions we create a 4-16 (using it as a 4-8 as the base implementation of the CPU uses 8 instruction) demux using one demux to decode the MSBs and 2 demux to decode the LSBs*
Question: *Which register is used as input to the demultiplexer's address bits*
Answer: *The Opcode register*
Question: *Based on what you know from the clock and the different phases generated, what is the common input to the demultiplexer ?*
Answer: *Phase 3 is the control signal that is demultiplexed based on the instruction opcode*
Question: *What is the encoding of the following instruction STAM (0100)?*
Answer: *0000 0000 0000 0000 0001 0000 0000 0000*
Question: *Consider the following:*

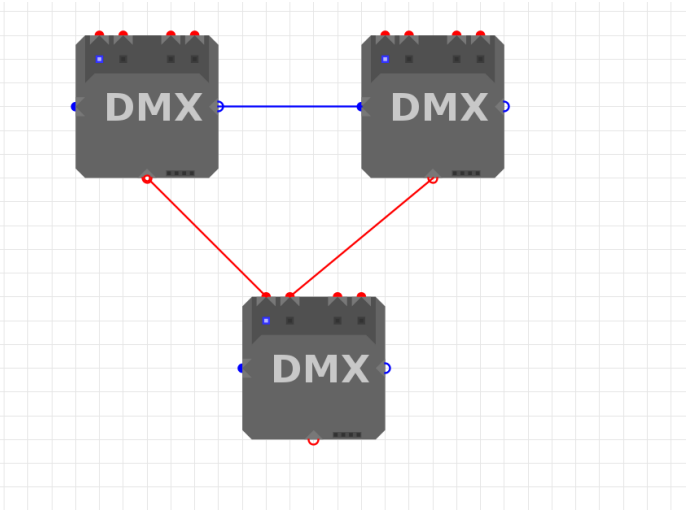

Figure 3: Demultplexers

1. *What is the purpose of this circuit ?* ***Answer:*** *One-hot encode the Opcode*

2. *What is the input the bottom and top demultiplexers ?* ***Answer:*** *Bottom DMX Opcode MSB (2-bit), Top DMX Opcode LSB (2-bit)*

3. *Which component in Modulesim can provide the desired functionality for (2.)?* ***Answer:*** *Split-Merge*

4.2.3 Control Signal

Different stages of the CPU either derive the same control signal or target the same hardware.
Answer: *We use OR blocks*
Question: *What ModuleSim components can you use to implement this functionality ?*
Question: *Consider the following ALU diagram and 4-bit control logic the each of the components:*

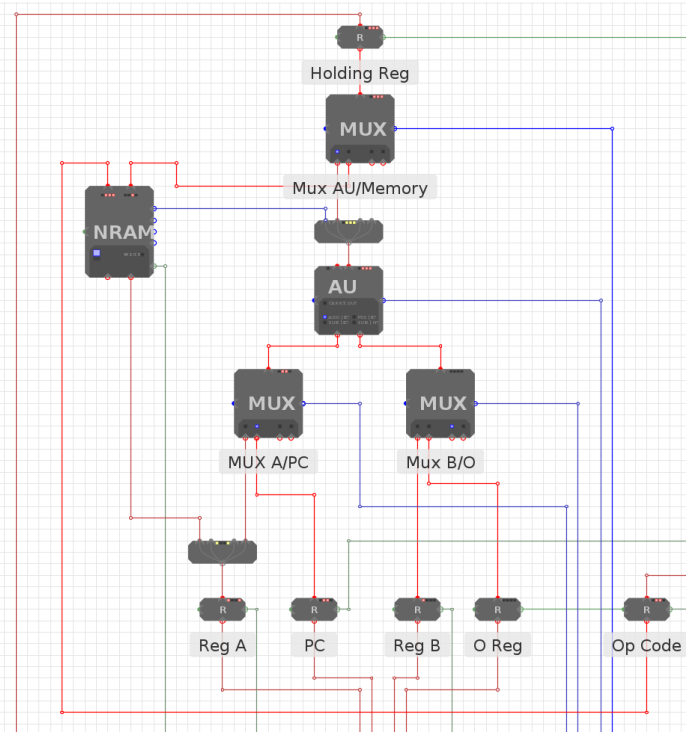

Figure 4: ALU

1. *What is the input to the AU/Memory MUX in phase 1 ?*
   Answer: *0001 to select the NRAM data*

2. *Which register(s) are "enabled" in phase 1.b ?*
   Answer: *O Reg, Opcode Reg*

3. *What are the inputs to MUX A/PC and B/O in phase 2 ?*
   Answer: *0001 for MUX A/PC to select the PC and 0010 for MUX B/O to select a '0' so we can perform PC + 0 + Carry (the correct input need to be set to AU)*

4. *Describe the process of implementing the STAM instructions. (i.e. Dataflow and control signal)* ***Answer:*** *The first step is to fetch the instruction (Refer to the xlsx file)*
   Then we need to incrememt PC+1
   Decode the instuction using our DMX system