Specification Document

Omkar Girish Kamath

October 13, 2022

Contents

1 Instruction set of the processor

In this instruction syntax X=Not used, K=Constant, A=Instruction Address, P=Data Address

Load ACC kk : 0000 XXXX KKKKKKKK

Add ACC kk : 0100 XXXX KKKKKKKK

And ACC kk : 0001 XXXX KKKKKKKK

Sub ACC kk : 0110 XXXX KKKKKKKK

Input ACC pp : 1010 XXXX PPPPPPPP

Output ACC pp : 1110 XXXX PPPPPPPP

Jump U aa : 1000 XXXX AAAAAAAA

Jump Z aa : 1001 00XX AAAAAAAA

Jump C aa : 1001 10XX AAAAAAAA

Jump NZ aa : 1001 01XX AAAAAAAA

Jump NC aa : 1001 11XX AAAAAAAA

The register transfer level (RTL) description of each of these instructions is:

Load ACC kk : ACC <- KK

Add ACC kk : ACC <- ACC + KK

And ACC kk : ACC <- ACC & KK

Sub ACC kk : ACC <- ACC - KK

Input ACC pp : ACC <- M[PP]

Output ACC pp : M[PP] <- ACC

Jump U aa : PC <- AA

Jump Z aa : IF Z=1 PC <- AA ELSE PC <- PC + 1

Jump C aa : IF C=1 PC <- AA ELSE PC <- PC + 1

Jump NZ aa : IF Z=0 PC <- AA ELSE PC <- PC + 1

Jump NC aa : IF C=0 PC <- AA ELSE PC <- PC + 1

Here '->' indicates updated with .

The processor has an extra cycle to save on hardware which would have been required for incrementing the PC . So the processor follows **fetch-decode-execute-increment** cycle .

2 Modules in the processor

2.1 RAM

2.1.1 Theory

2.1.2 Interface

2.2 Program Counter

2.2.1 Theory

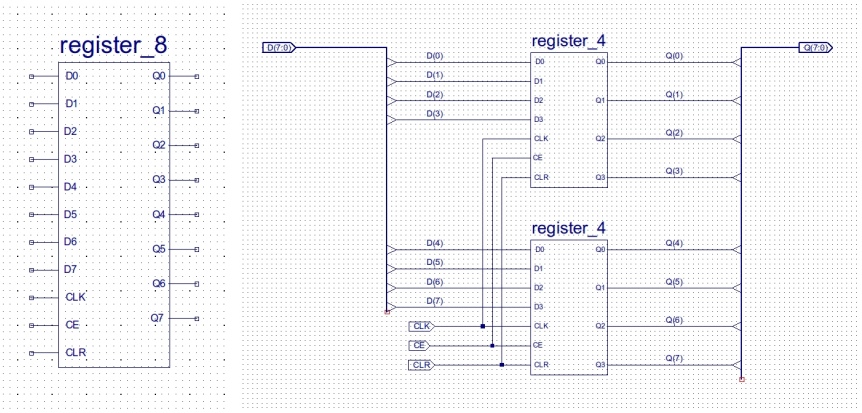


Figure 1:

8 bit register used to store the address **current** instruction being executed. It is incremented after every *fetch-decode-execute-increment* cycle .

2.2.2 Interface

```
module pc (
d,
clk,
ce,
clr,
q
);
```

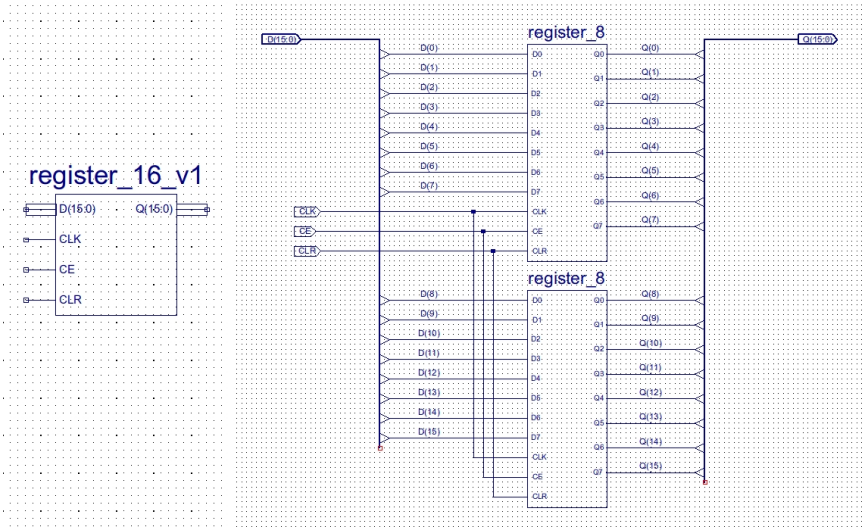
input [7:0] d ;
input clk ;
input ce ;
input clr ;
output [7:0]q;

signal name	type	size
d	input	8 bits
clk	input	1 bit
ce	input	1 bit
clr	input	1 bit
q	output	8 bits

2.3 Instruction Register

2.3.1 Theory

Instruction Register (IR) : **16** bit register, updated at the end of the fetch phase with the instruction to be processed (decoded and executed).



2.3.2 Interface

module ir (
d,

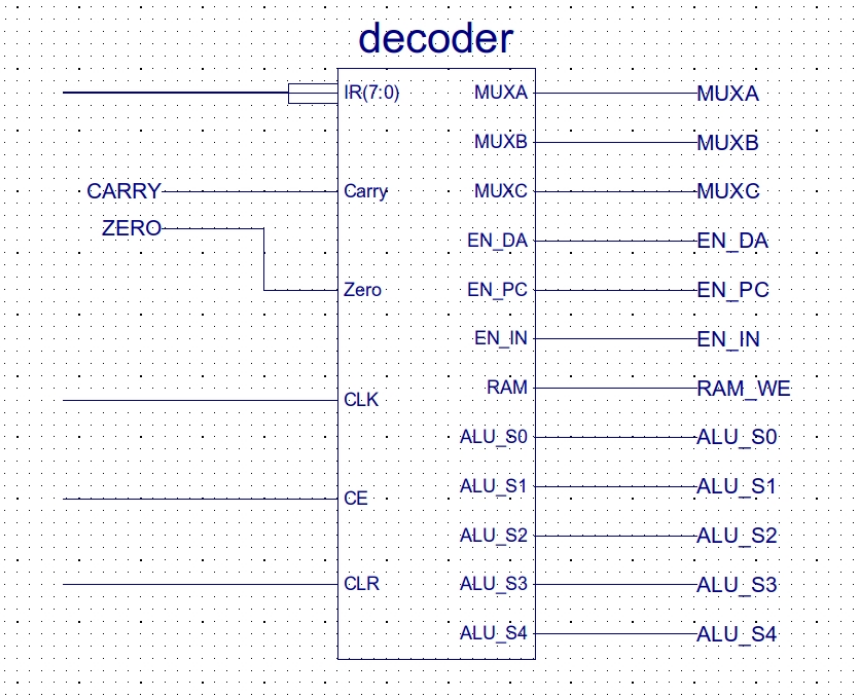
```
clk,
ce,
clr,
q
);

input [15:0] d ;
input clk ;
input ce ;
input clr ;
output [15:0] q ;
```

signal name	type	size
d	input	16 bits
clk	input	1 bit
ce	input	1 bit
clr	input	1 bit
q	output	16 bits

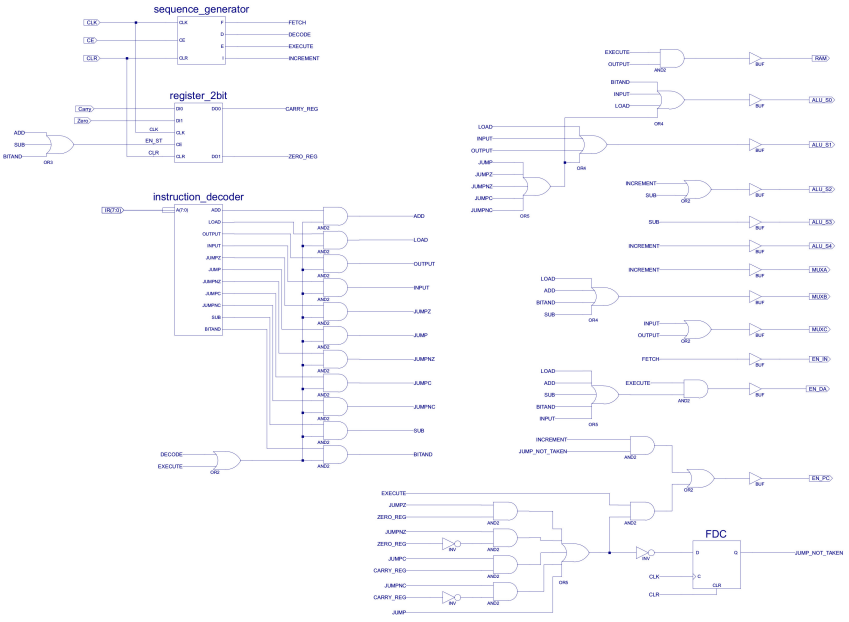
2.4 Decoder

2.4.1 Theory

It generates the sequence of control signals needed to perform the functions defined by each instruction by considering the current state of the processor and the current instruction . These are contained within the decoder or control-logic block , the circuit diagram symbol for this component is shown.



The different parts of the decoder is shown in the figure given



(a) SEQUENCE GENERATOR

To identify which phase the processor is in a sequence generator is used , This is a simple ring counter , using a one-hot (Link) encoded value to indicate the processor's state . Initially the value 1000 is loaded into the counter (fetch code), on each clock pulse the one-hot bit is then moved along the flip-flop chain, looping back to the start after four clock cycles. To determine the processor's state you simply identify which bit position is set to a logic 1. One-hot encoding i.e. when a number only ever has one bit set, is very easy to decode in hardware, but its sparse encoding means you need a lot of bits to represent larger values as not all bit states are used. A four bit one-hot value can represent 4 states, or the value 0 to 3, using a binary encoding you could represent the value 0 to 15, but you would need to decode all bits to determine its value, where as using one-hot you only need to look at one bit.

1000 : Fetch

0100 : Decode

0010 : Execute

0001 : Increment

(b) INSTRUCTION DECODER

The top 4 to 6 bits of each instruction defines the opcode: an unique binary pattern that allows the CPU to identify what function needs to be performed, where the data (operands) are and where any result produced should be stored. Note, the top nibble (4bit) of each instruction is unique to that instruction. The instruction decoder converts the unique opcode into a one-hot value, these are then used during the Decode and Execute phases to control the processor's hardware. To ensure these signals are not active during the Fetch and Increment phases they are ANDed (zeroed) with the result of the logical OR of the Decode and Execute signals from the sequence generator.

Most of this control logic is quite intuitive, you simply combine the one-hot output from the Decoder with the state bits from the Ring-counter to produce the logic 1's in each row of the table in figure 32 . A slightly more complex bit is the Jump logic . If the processor is in the Execute phase, the instruction decoder and status signals determine if the program counter (PC) should be updated i.e. should the jump address be loaded into the PC. If a JUMP instruction is taken, then the system does not need to increment

the PC, as it already contains the address of the next instruction. Therefore, when the processor is in the Increment phase it checks to see if a jump has been taken, if it has been the PC is not enabled i.e. the result PC+1 is not stored in the program counter.

2.4.2 Interface

Name	size	function	type
mux_a	1	ALU A input MUX control	output
mux_b	1	ALU B input MUX control	output
mux_c	1	address MUX control, selecting PC or IR	output
en_{da}	1	accumulator (ACC) register update control	output
en_{pc}	1	program counter (PC) register update control	output
en_{ir}	1	instruction register (IR) update control	output
ram_{we}	1	memory write enable control	output
alu_c	5	ALU control line	output
ir	8	high byte of instruction register, contains opcode	input
zero	1	connected to ALU output, if 1 indicates result is zero	output
clk	1	system clock	input
ce	1	clock enable, normally set to 1, if set to 0 processor will HALT	input
clr	1	system reset, if pulsed high system will be reset	input

MUX_A : output, ALU A input MUX control

MUX_B : output, ALU B input MUX control

MUX_C : output, address MUX control, selecting PC or IR

EN_{DA} : output, accumulator (ACC) register update control

EN_{PC} : output, program counter (PC) register update control

EN_{IR} : output, instruction register (IR) update control

RAM_{WE} : output, memory write enable control

ALU_{S0} : output, ALU control line

ALU_{S1} : output, ALU control line

ALU_{S2} : output, ALU control line

ALU_{S3} : output, ALU control line

ALU_{S4} : output, ALU control line

(**combining all the ALU control lines we get a 5 bit out alu_c signal**)

IR : input bus, 8bits, high byte of instruction register, contains opcode

ZERO : input, driven by 8bit NOR gate connected to ALU output, if 1 indicates result is zero

CARRY : input, driven by carry out (Cout) of ALU

CLK : input, system clock

CE : input, clock enable, normally set to 1, if set to 0 processor will HALT

CLR : input, system reset, if pulsed high system will be reset

2.5 Accumulator

2.5.1 Theory

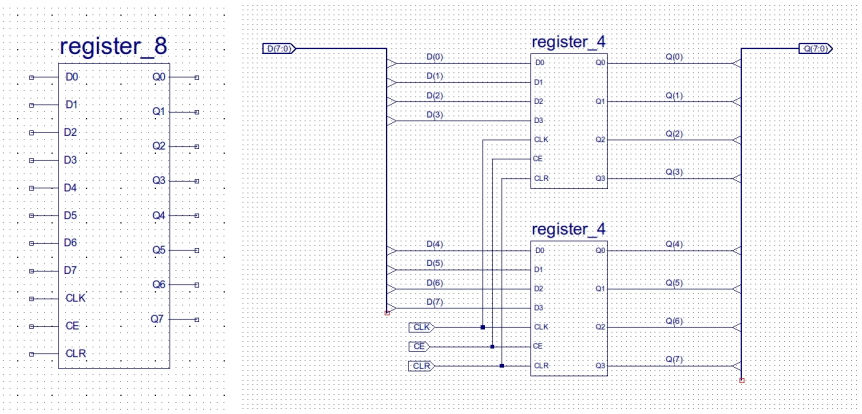


Figure 2:

Accumulator (ACC) : *8* bit register, a general purpose data register, providing data (operand) to be processed by the ALU and used to **store** any result produced. Note, we can only store one 8 bit value at a time on the processor, other data values will need to be buffered in external memory.

2.5.2 Interface

module pc (
d,
clk,

```
ce,
clr,
q
);

input [7:0] d ;
input clk ;
input ce ;
input clr ;
output [7:0] q;
```

signal name	type	size
d	input	8 bits
clk	input	1 bit
ce	input	1 bit
clr	input	1 bit
q	output	8 bits

2.6 ALU

2.6.1 Theory

2.6.2 Interface

2.7 MUX

2.7.1 $\mathbf{MUX_{IR\ to\ ALU}}$

2.7.2 $\mathbf{MUX_{PC\ to\ ALU}}$

2.7.3 $\mathbf{MUX_{Address\ in\ RAM}}$