 [Translated from Greek to English - www.onlinedoctranslator.com](https://www.onlinedoctranslator.com/en/?utm_source=onlinedoctranslator&utm_medium=docx&utm_campaign=attribution)

3/15/2018

# 2nd Laboratory Exercise

**DESIGNING THE BASIC STEPS OF A SIMPLE PROCESSOR'S DATAPATH**

**11/03/2018**

Group LAB31235515

|  |
| --- |
| *CHRISTOS ZISKAS 2014030191* |
| *ANTONIS ANTONAKAKIS 2014030160* |

## Purpose of laboratory exercise

It is the friction with language to define command architecture as an object of study. At the same time, here is the design of the stages that make up parts of a simple processor and concern stages for the following functions:

* Recall of orders
* Command decoding
* Execution of commands
* Memory access

The implementation of the entire design features the register file and the arithmetic unit (ALU) combined with additional memory elements as well as combinational logic

## Preparation

Correlations regarding the instruction set architecture of a non-pipelined processor based on a subset of the CHARIS-4 instruction set architecture are presented.

The instruction set architecture consists of:

* 32 registers of 32 bits. The R0 register is always zero.
* 32 bit wide instructions with size and position fields described  
  below.
* Arithmetic and logical operations commands: add, sub, and, not, or, shr, shl,  
  sla, rol, ror, li, addi, andi, ori.
* Branch commands: b, beq, bneq.
* Memory commands: lb, sb, lw, sw.

The above commands have two format types:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 6-bits | 5-bits | 5-bits | 5-bits | 5-bits | 6-bits |
| Opcode | Rs | rd | rt | not-used | func |

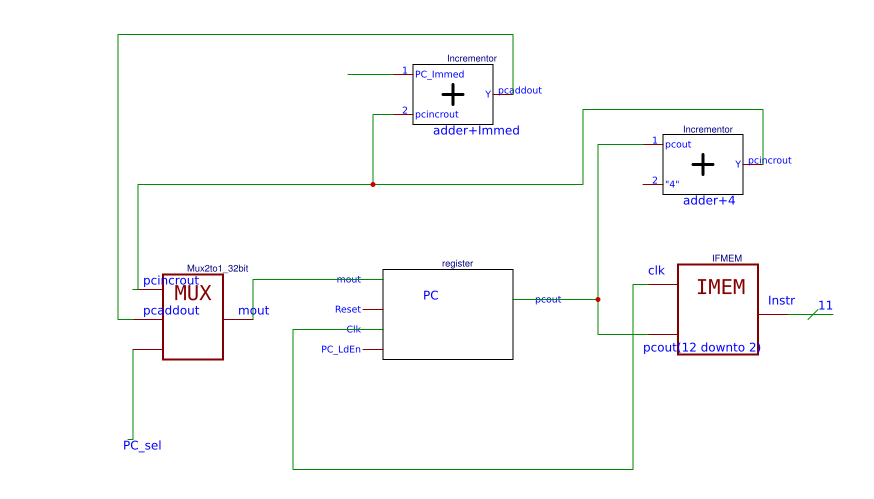
|  |  |  |  |
| --- | --- | --- | --- |
| 6-bits | 5-bits | 5-bits | 16-bits |
| Opcode | Rs | rd | Immediate |

At the same time, schematic diagrams of the tiers are presented separately, to illustrate their functionality which contributes to the partial implementation of the datapath while it is completed in combination with combinatorial logic objects.

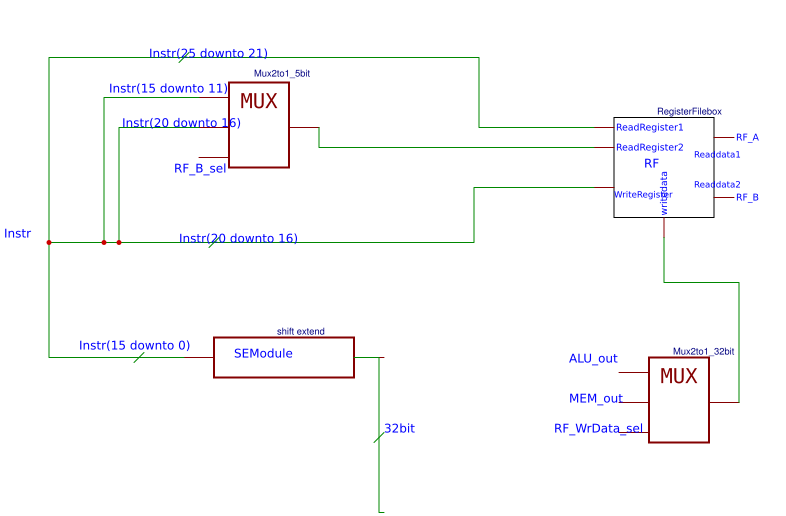
Codification of commands

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode | FUNC | MANDATE | ACT |
| 100000 | 110000 | add | RF[rd] ← RF[rs] + RF[rt] |
| 100000 | 110001 | sub | RF[rd] ← RF[rs] - RF[rt] |
| 100000 | 110010 | nand | RF[rd] ← RF[rs] NAND RF[rt] |
| 100000 | 110100 | not | RF[rd] ← ! RF[rs] |
| 100000 | 110011 | or | RF[rd] ← RF[rs] | RF[rt] |
| 100000 | 111000 | Ms | RF[rd] ← RF[rs] >>1 |
| 100000 | 111001 | sll | RF[rd] ← RF[rs] <<1 (Logical, zero fill LSB) |
| 100000 | 111010 | srl | RF[rd] ← RF[rs] >>1 (Logical, zero fill MSB) |
| 100000 | 111100 | rol | RF[rd] ← Rotate left(RF[rs]) |
| 100000 | 111101 | ror | RF[rd] ← Rotate right(RF[rs]) |
| 111000 | - | li | RF[rd] ← SignExtend(Imm) |
| 111001 | - | lui | RF[rd] ← Imm << 16 (zero-fill) |
| 110000 | - | addi | RF[rd] ← RF[rs] + SignExtend(Imm) |
| 110010 | - | nandi | RF[rd] ← RF[rs] NAND ZeroFill(Imm) |
| 110011 | - | ori | RF[rd] ← RF[rs] | ZeroFill(Imm) |
| 111111 | - | b | PC ← PC + 4 + (SignExtend(Imm) << 2) |
| 000000 | - | beq | if (RF[rs] == RF[rd]) PC ← PC + 4 + (SignExtend(Imm) << 2) else PC ← PC + 4 |
| 000001 | - | bne | if (RF[rs] != RF[rd]) PC ← PC + 4 + (SignExtend(Imm) << 2) else PC ← PC + 4 |
| 000011 | - | lb | RF[rd] ← ZeroFill(31 downto 8) & MEM[RF[rs] + SignExtend(Imm)](7 downto 0) |
| 000111 | - | sb | MEM[RF[rs] + SignExtend(Imm)] ← ZeroFill(31 downto 8) & RF[rd] (7 downto 0) |
| 001111 | - | lw | RF[rd] ← MEM[RF[rs] + SignExtend(Imm)] |
| 011111 | - | sw | MEM[RF[rs] + SignExtend(Imm)] ← RF[rd] |

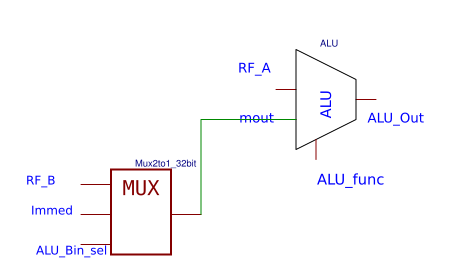
Instruction recall stage (IF)



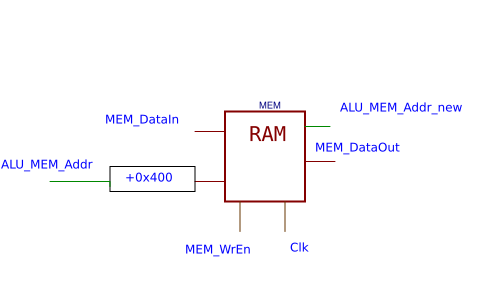
Stepcommand decoding (DECODE)



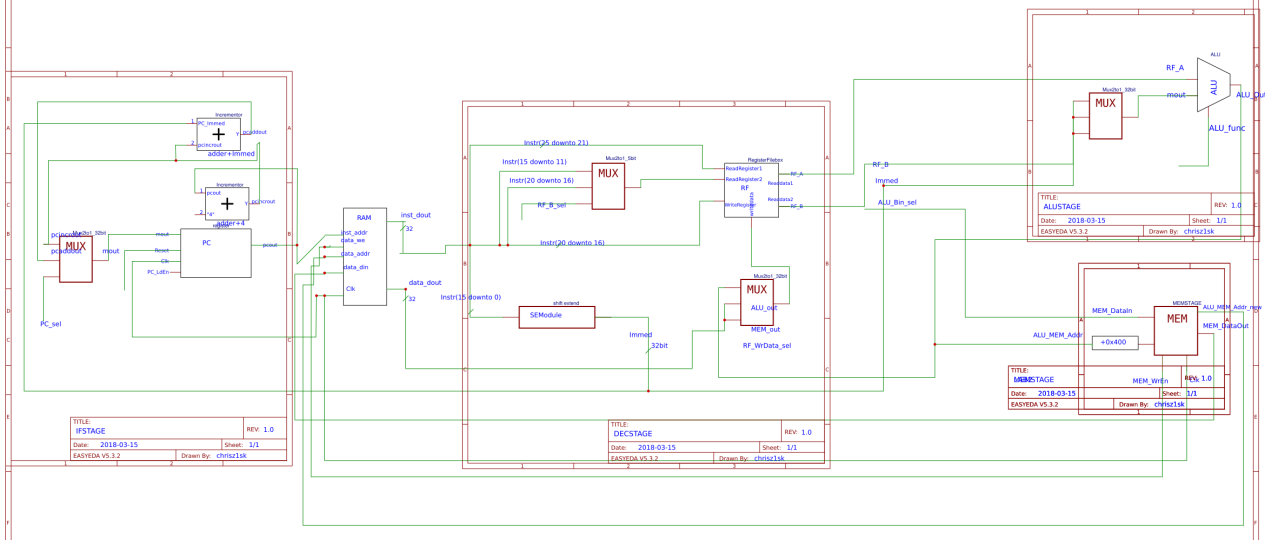
Instruction Execution Unit (ALU)



Level Access Memory (MEM)



Total Tier (Datapath)



## Description

The issue of command architecture concerns the encoding of CHARIS, observing their grouping for the easiest possible decoding to produce the control signals of the integrated system.

As far as the IF stage is concerned, it contributes to the organization of memory according to the CHARIS instructions while at the same time arranging the branch instructions. It consists of the program counter register, adders and a multiplexer that in the branch commands guides the memory according to the value assigned to Immediate. The value of Immidiate is slipped in these instructions for proper memory management. Otherwise the memory address is incremented by one location and that updates the register (+4 bits = 1 memory location).

Next, the DECODE stage is managed by a register file, multiplexers and a module that converts the immidiate into its appropriate encoding for completing the architecture instructions. The data input multiplexer deals with commands that either store and take data from memory and update it for writing, or any other commands that come in from the output of the ALU. The address registers depend on the commands that are executed and are configured by the format of the commands (the controls of the multiplexes are activated/deactivated). For R-type commands, disable the control of the multiplexer at the input of the register file so that the values ​​of the registers rs,rt interact and are stored in rd. The coding of immidiate is done according to the command architecture.

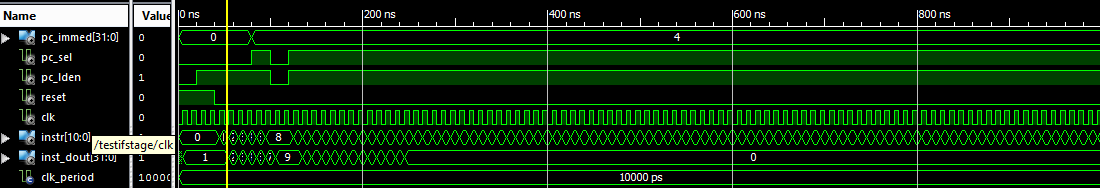
In addition, according to the ALU stage that has the ALU calculation unit as well as a multiplexer, the results arranged by the ALU are output. The control of the ALU affects the calculations (shift, logic, arithmetic) while the control of the multiplexer affects the input and concerns the immidiate input to the result due to each command (it comes from the immidiate coding unit)

The design is completed with the MEM stage. The actual address arriving in memory is the output from the ALU plus 0x400 - the MEM stage accepts this as the address. Memory data enters from the MEM stage and is directed to RAM. It essentially serves as an intermediate step for data to be driven into RAM

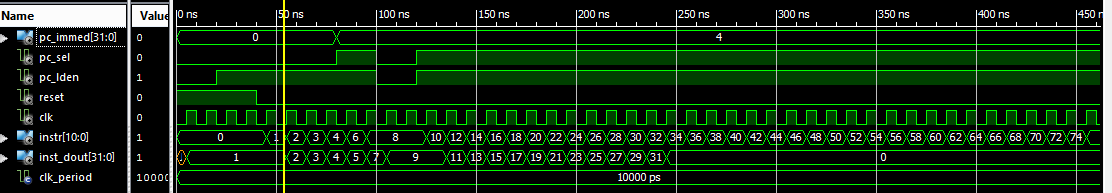
## Waveforms-Simulation

The waveforms of the various stages are shown

Command Recall Tier (IF Unit)

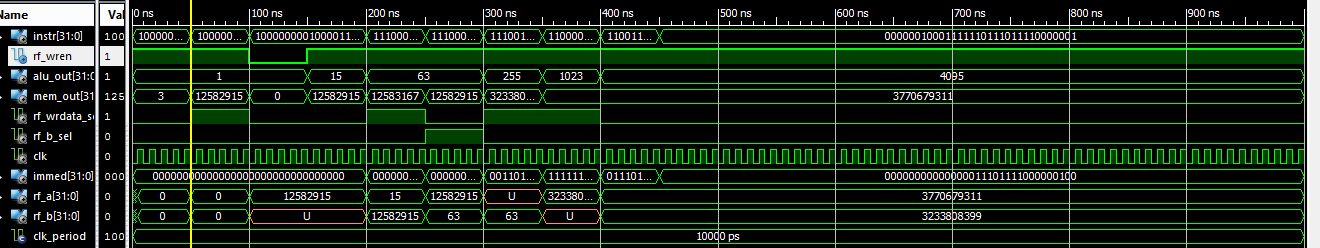


The input received by the IF stage allows the functionality of the design. The memory starts from the zero position and the system waits for the reset signal to be deactivated. With reset & enable the system remains inactive. The reset "falls" and the memory is addressed. Each time an instruction is executed outside a branch , the memory is extended by one location (+4 bits). When executing branch commands the memory is additionally increased according to the address of the immidiate. The enable is disabled and the tier is deactivated.

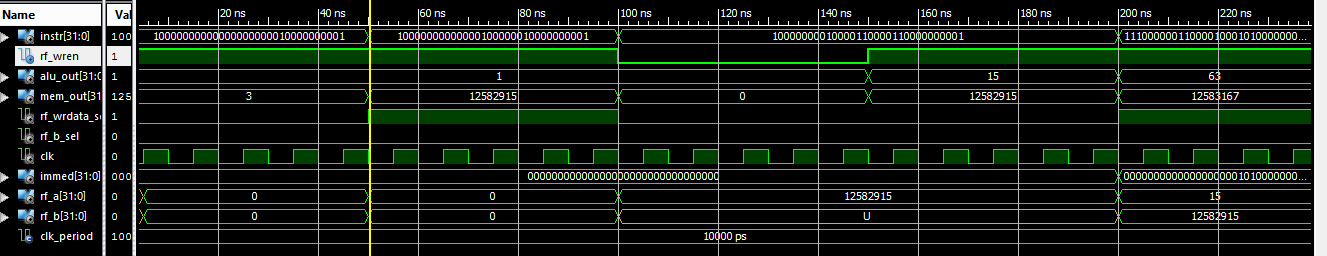


Enable is activated and the operation of the circuit continues considering that we accept branch commands and the value of the memory increases by the value of Immidiate - sliding it by 2 positions so that we refer to memory positions - always referring to word addressable memory. So the design ends up reading the entire input file without and with memory branching.

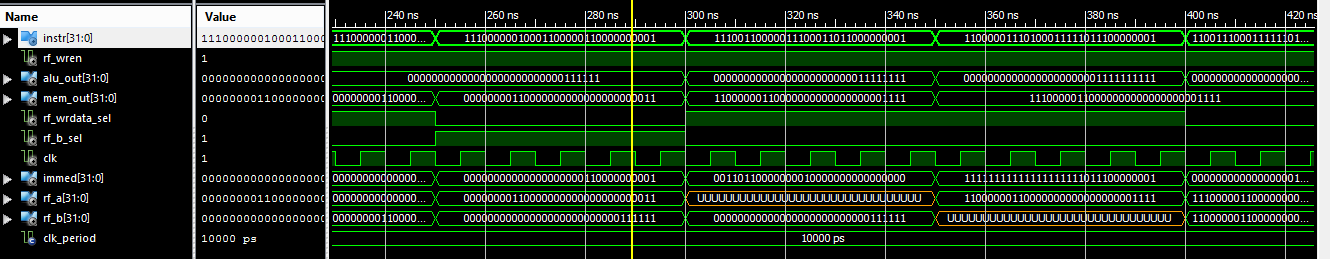
Command Decoder Stage (DEC Unit)



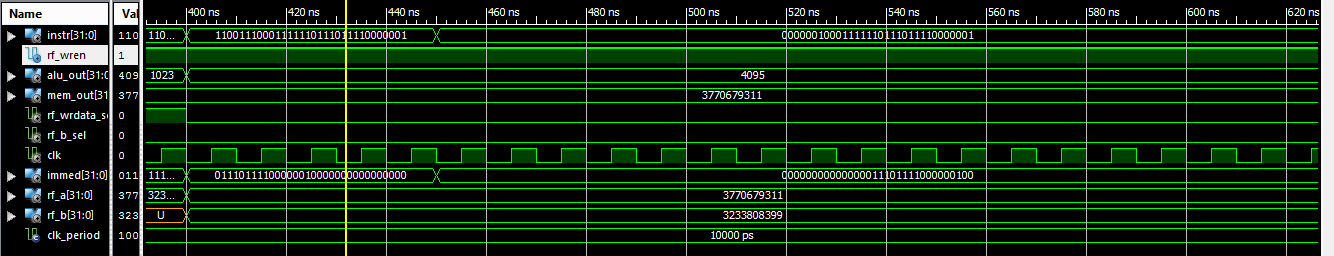
This part of the design decodes the instruction fetched from memory. The command is split into read & write, Opcode, Immediate chunks and the input data is written to the register file.



The initial read addresses refer to the zero register to which it is impossible to write. Zero output appears as the output while Immidiate depends on the type of commands implemented (Rtype, Itype , Store, branch, Load). For R-type Immediate is zero. Only the write address is different, so register 2 is written with the value coming from MEM. It depends on the multiplexer control of the input data. For 0 the result comes from the ALU otherwise from the memory. As a continuation, register 2 and 1 are read while register 3 is written. Enable is disabled so there is no writing. It activates and continues. The previous address is entered as read input and the record is verified.

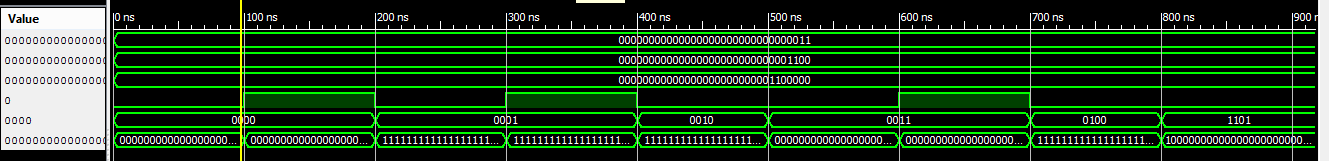


Test for type I commands continues. immidiate accepts sign extend while the write comes from memory. Writes continue and are progressively verified while immidiate conversions are also presented according to the opcode displayed by the command. Register 3 is verified and register 6 is written by the ALU. The input multiplexer control when active helps pass write and read address#2. Values ​​are stored in the various registers while at the same time the values ​​received by immidiate are checked



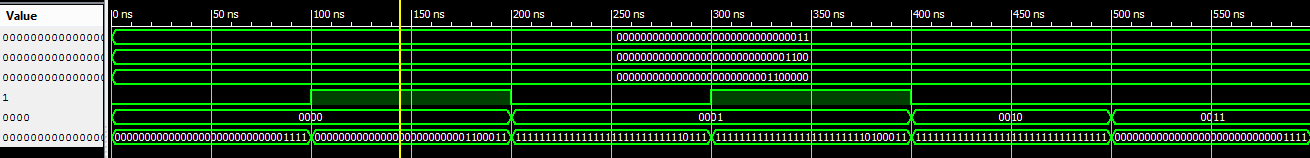
|  |  |  |
| --- | --- | --- |
| Opcode | Instr-16bit | Immediate-32 bit |
| 111000 (SignExtend) | 0000110000000001 | 00000000000000000000110000000001 |
| 111001 (Imm<<16(zerofill)) | 0011011000000001 | 00110110000000010000000000000000 |
| 110000 (SignExtend(Imm)) | 1111011100000001 | 11111111111111111111011100000001 |
| 110011 (ZeroFill(Imm)) | 0111011110000001 | 00000000000000000111011110000001 |
| 000000SignExtend(Imm)<<2 | 0111011110000001 | 00000000000000011101111000000100 |
| 100000 | X | 00000000000000000000000000000000 |

Instruction Execution Stage (ALU Unit)

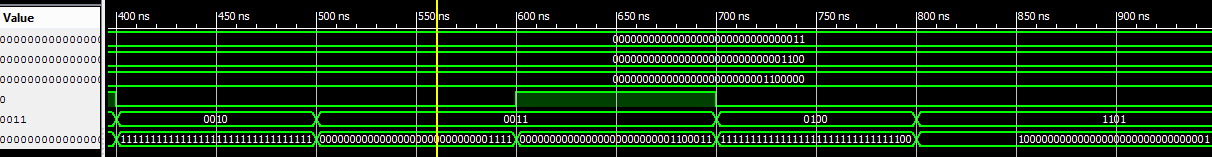


The ALU stage specializes in the recognition of operations and additionally configures the selection of operands by examining the corresponding signals (Alu\_Bin\_Sel). For Alu\_bin\_sel = 0, the operands are the outputs from rf while for a value of 1 the second operant is configured by immidiate. The ALU performs its functionality according to the controls it has and additionally performs the operations according to the func.

It is observed that first the outputs from rf are added and then rfa is added with Immidiate. Then the act of subtraction is carried out in a similar sequence.



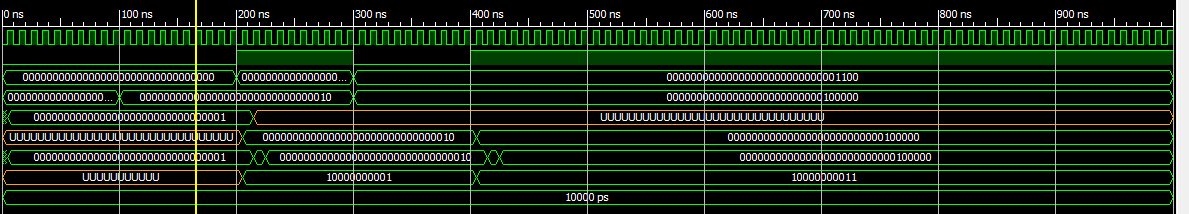
The overall design is verified by applying logic and sliding operations.



|  |  |  |
| --- | --- | --- |
| 0001 | ...000011- ..001100 | 11111111111111111111111111110111 |
| 0010 | ...000011 NAND ..001100 | 11111111111111111111111111111111 |
| 0011 | ...000011 OR ..001100 | ..001111 |
| 0100 | 000000000000000000000000000000011 | 11111111111111111111111111111100 |
| 1101 | 000000000000000000000000000000011 | 100000000000000000000000000000001 |
| 0000 | ...000011+..001100 | ..001111 |

Memory access level (MEM Unit)

The MEM layer behaves as an intermediate storage medium. For enable the input of the data passes through the stage to register the data in the ram



And the output address to be set as exported. To disable-enable enable the memory repeats data recording.

## Conclusions/Problems

The lab exercise leads to significant results as the processor implementation continues to progress by analyzing instruction processing tiers while looking at segmentation and memory management for a single-cycle datapath