Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)

Department of CSE

Programme: B.E **Term: Jan to May 2019**
Course: Computer Organization **Course Code: CS45**

Activity V: Designing an ALU to perform arithmetic and logical functions using Logisim simulator.

Name:PRAJWAL A	**Marks: /10**	**Date:20-05-2020**
USN:1MS18CS092	**Signature of the Faculty:**	

Objective: To simulate the working of Arithmetic and Logical Unit using simulator.

Simulator Description: Logisim is an educational tool for designing and simulating digital logic circuits. With its simple toolbar interface and simulation of circuits as you build them, it is simple enough to facilitate learning the most basic concepts related to logic circuits. With the capacity to build larger circuits from smaller sub circuits, and to draw bundles of wires with a single mouse drag, Logisim can be used (and is used) to design and simulate entire CPUs for educational purposes.

Activity to be performed by students:

list out the Steps in desiging ALU

Step 1: Add the two input pins

Step 1: Drop two East facing input pins on the Canvas 4 bits each Label A and B ensure that each input is 4 bits.

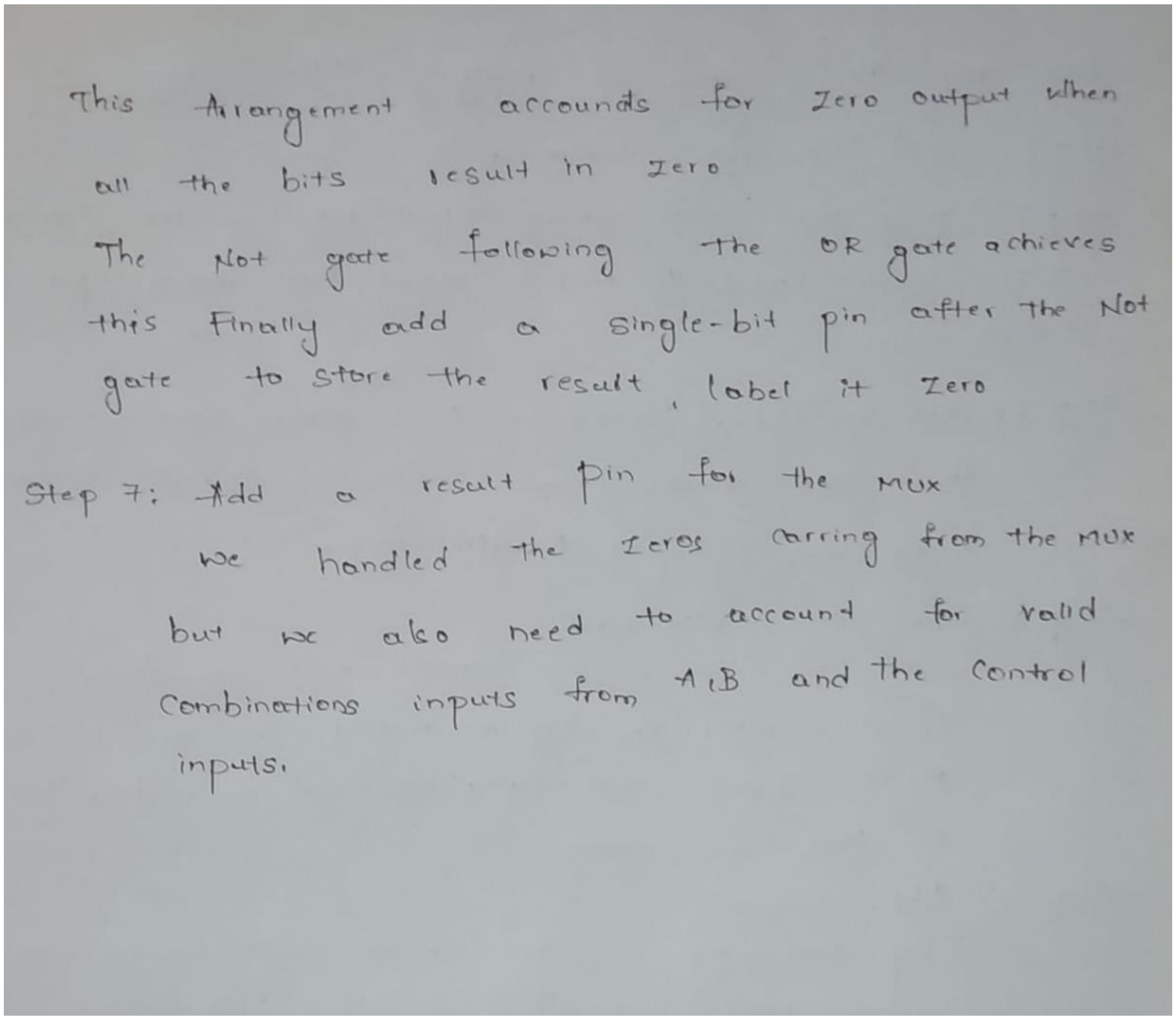
Step 2: Add the Adder | Subtractor and gates Now we add the Sub circuits Created earlier. Select circuits under main project handler folder

Step 3: Add the Multiplexers
Take on or more inputs and generate a Single Output In logisim multiplexers are under plexer folder Click Multiplexer icon and drop two of them into canvas

Step 4: Add controls
Drop two pins on the canvas north facing with 1 data bit. Label them 0 and 1 respectively

Step 5. Add a splitter
Next we add a Splitter into our circuit that takes one line from the Second multiplexer and Split to 4 inputs to an OR gate for a 4 bit ALU

Step 6: Add another OR gate and Not gate Now we add an OR gate after the Splitter which has 4 inputs. To right of the OR gate and a NOT Gate



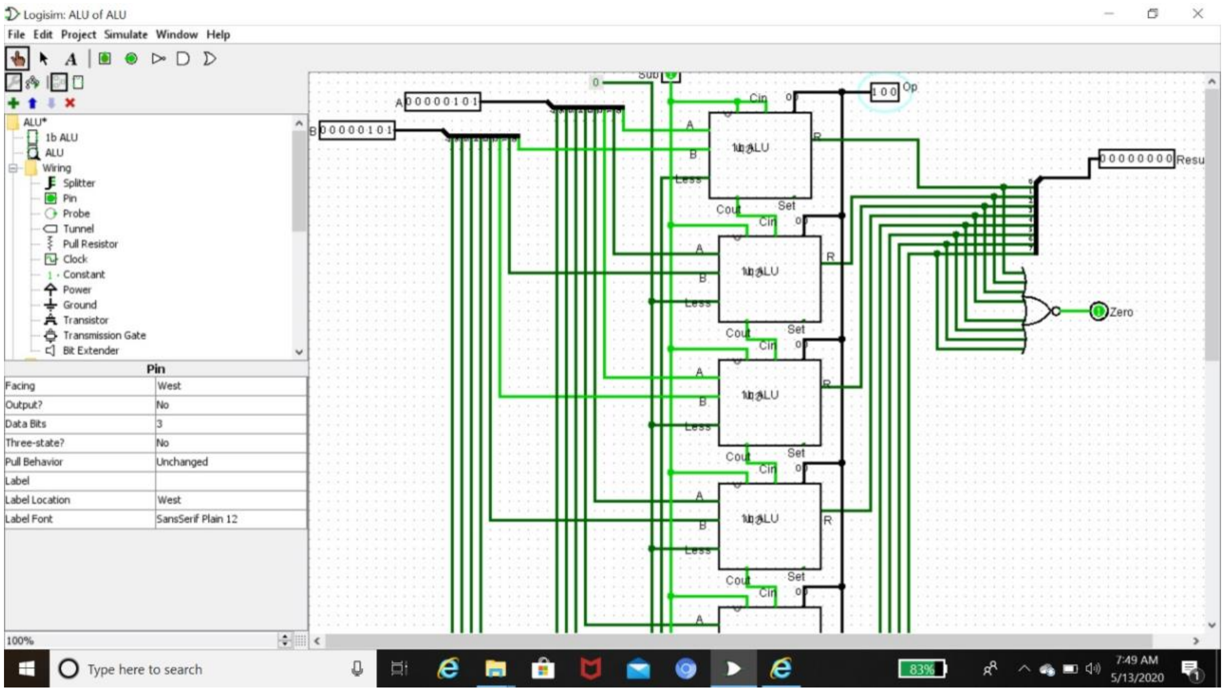
This Arrangement accounds for Zero output when all the bits result in Zero

The Not gate following the OR gate achieves this Finally add a single-bit pin after the Not gate to store the result, label it Zero

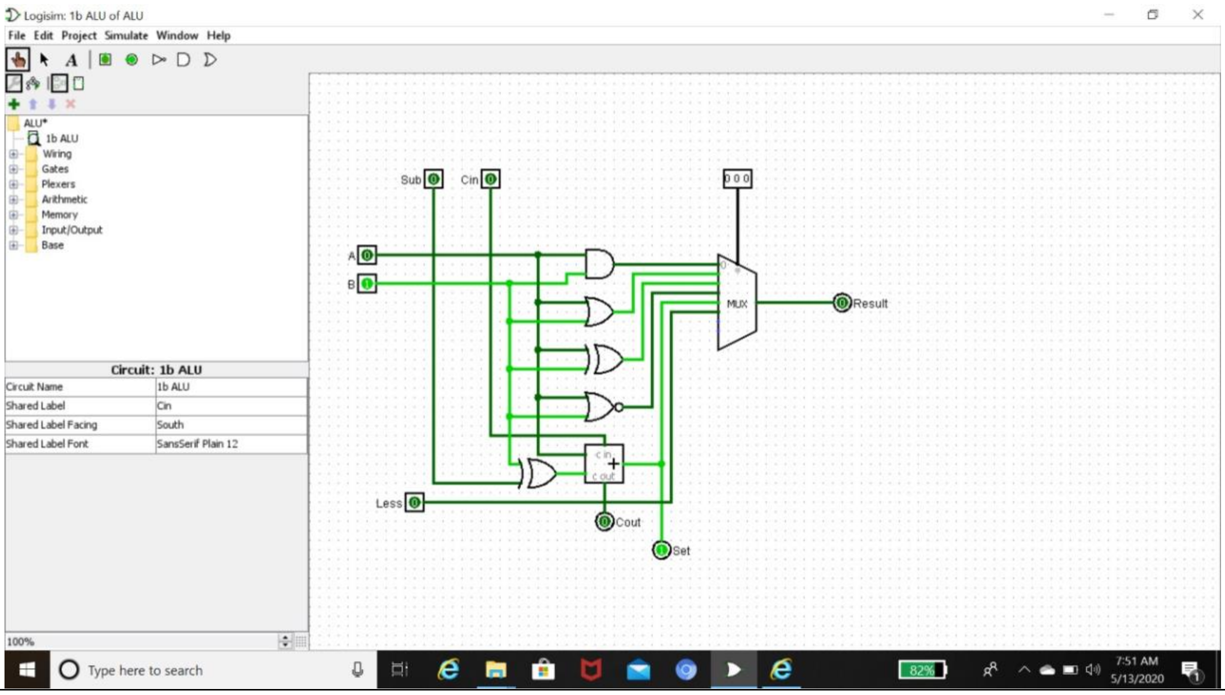
Step 7: Add a result pin for the MUX

we handled the Zeros carring from the MUX but we also need to account for valid Combinations inputs from A,B and the Control inputs.

SNAPSHOTS



Pin	
Facing	West
Output?	No
Data Bits	3
Three-state?	No
Pull Behavior	Unchanged
Label	
Label Location	West
Label Font	SansSerif Plain 12



Circuit: 1b ALU

Circuit Name	1b ALU
Shared Label	Cin
Shared Label Facing	South
Shared Label Font	SansSerif Plain 12

Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)

Department of CSE

Programme: B.E **Term: Jan to May 2019**
Course: Computer Organization **Course Code: CS45**

Activity VI: Designing memory system using Logisim simulator.

Name: PRAJWAL A	**Marks: /10**	**Date:20-05-2020**
USN: 1MS18CS092	**Signature of the Faculty:**	

Objective: To simulate the writing operation on memory.

Simulator Description: Logisim is an educational tool for designing and simulating digital logic circuits. With its simple toolbar interface and simulation of circuits as you build them, it is simple enough to facilitate learning the most basic concepts related to logic circuits. With the capacity to build larger circuits from smaller sub circuits, and to draw bundles of wires with a single mouse drag, Logisim can be used (and is used) to design and simulate entire CPUs for educational purposes.

Activity to be performed by students:

List out the steps in designing memory system

Step 1: Add RAM
Select a Seperate load and Store operation
for RAM

Step 2: Add Counter

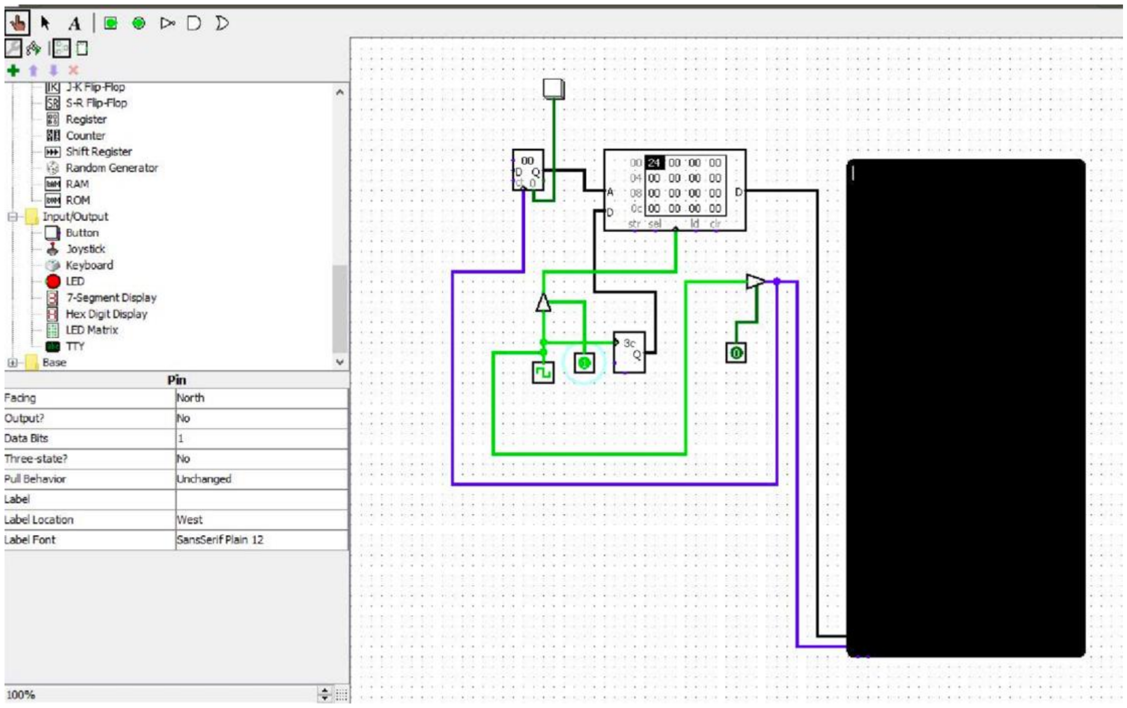
Step 3: Connect Counter clock, and Controlled
Buffer to the RAM

Step 3: Add TTY
To display Data Read on Memory

Step 4: Add Random Generator
To Generate different address location
Add input and another Controlled Buffer
to the Random Generator

Step 6: Add Button
Connect Button to Counter

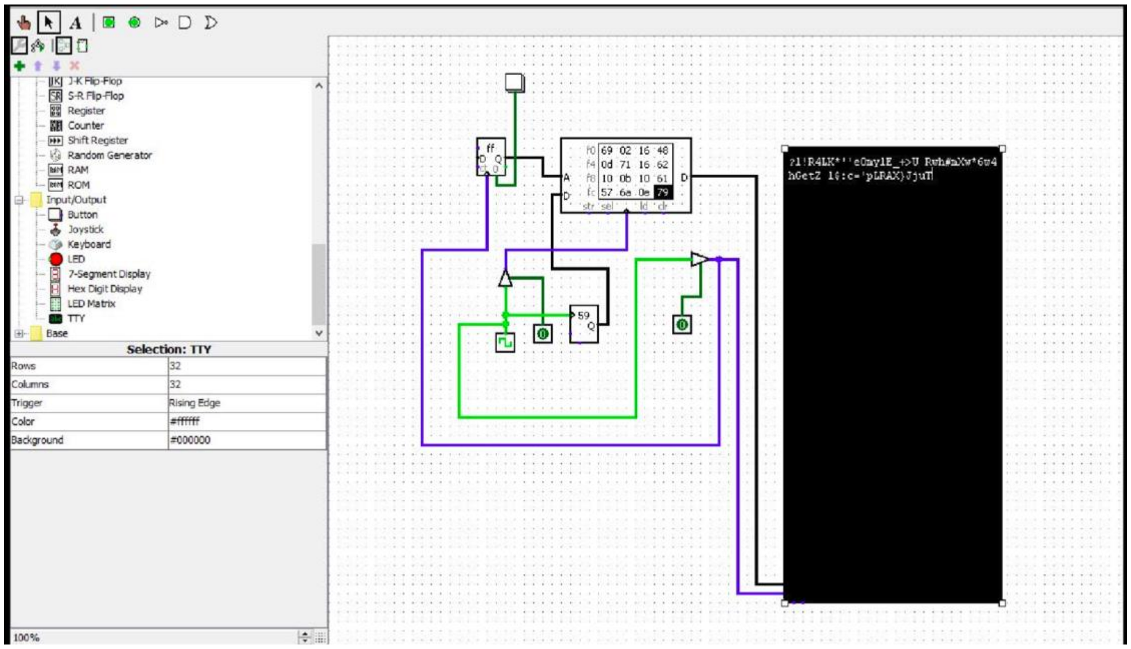
Observations and Snapshots:



Pin

Facing	North
Output?	No
Data Bits	1
Three-state?	No
Pull Behavior	Unchanged
Label	
Label Location	West
Label Font	SansSerif Plain 12

```
00 24 00 00 00
04 00 00 00 00
08 00 00 00 00
0c 00 00 00 00
```



Selection: TTY

Rows	32
Columns	32
Trigger	Rising Edge
Color	#ffffff
Background	#000000

```
f0 69 02 16 48
f4 0d 71 16 62
f8 10 0b 10 61
fc 57 6a 0e 79
```

```
?l!R4LK*''eOmylE_+>U Rwh#mXw*6w4
hGetZ l$:c='pLRAX)JjuT
```

Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)

Department of CSE

Programme: B.E **Term: Jan to May 2019**
Course: Computer Organization **Course Code: CS45**

Activity VII: To simulate advantages of using pipeline technique in executing a program.

Name:PRAJWAL A	**Marks: /10**	**Date:22-05-2020**
USN:1MS18CS092	**Signature of the Faculty:**	

Objective: To learn and analyze the performance of the CPU by overlapping of instructions using CPUOS-SIM simulator.

Simulator Used: CPUOS-SIM is a software development environment for the simulation of simple computers. It was developed by Dale Skrien to help users to understand computer architectures.

Modern CPU’s contain several semi-independent circuits involved in decoding and executing each machine instruction. Separate circuit elements perform each of these typical steps:

- Fetch the next instruction from memory into an internal CPU register.
- Decode the instruction to determine which function sub-circuits it requires.
- Read any input operands required from high-speed registers or directly from memory.
- Execute the operation using the selected sub-circuits.
- Write any output results to high-speed registers or directly to memory.

Separate sections of the CPU circuitry are used for each of these steps. This allows these circuit sections to be arranged into a sequential pipeline, with the output of one step feeding into the next step.

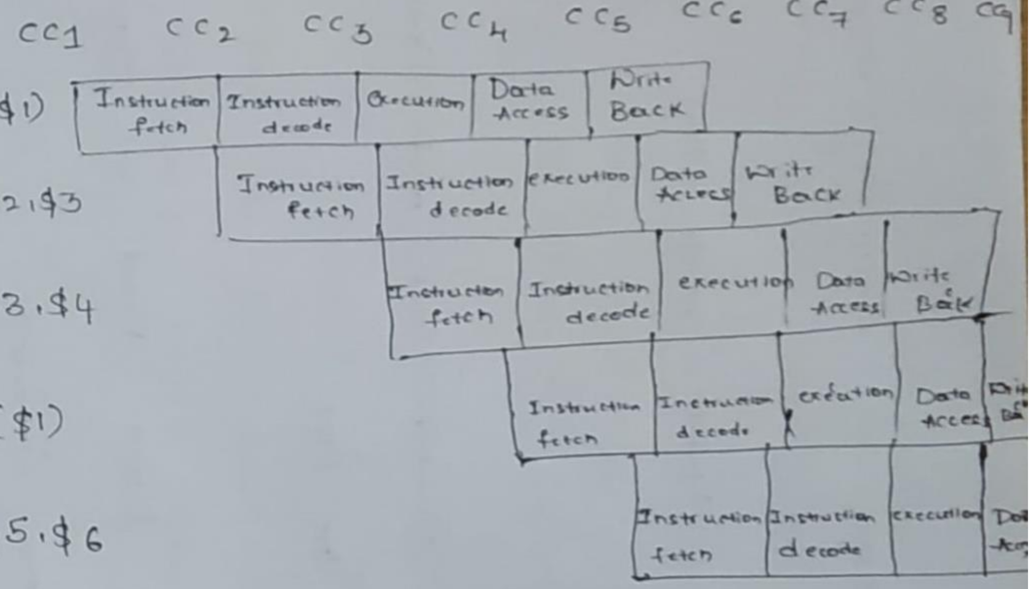
With diagram demonstrate the execution of the following instructions using pipelining technique.

```
lw   $10,20($1)
sub   $11, 42, $3
add  $12, $3, $4
lw   $13, 24($1)
add  $14, $5, $6
```

Time (in clock cycles)

Program execution order

in instruction



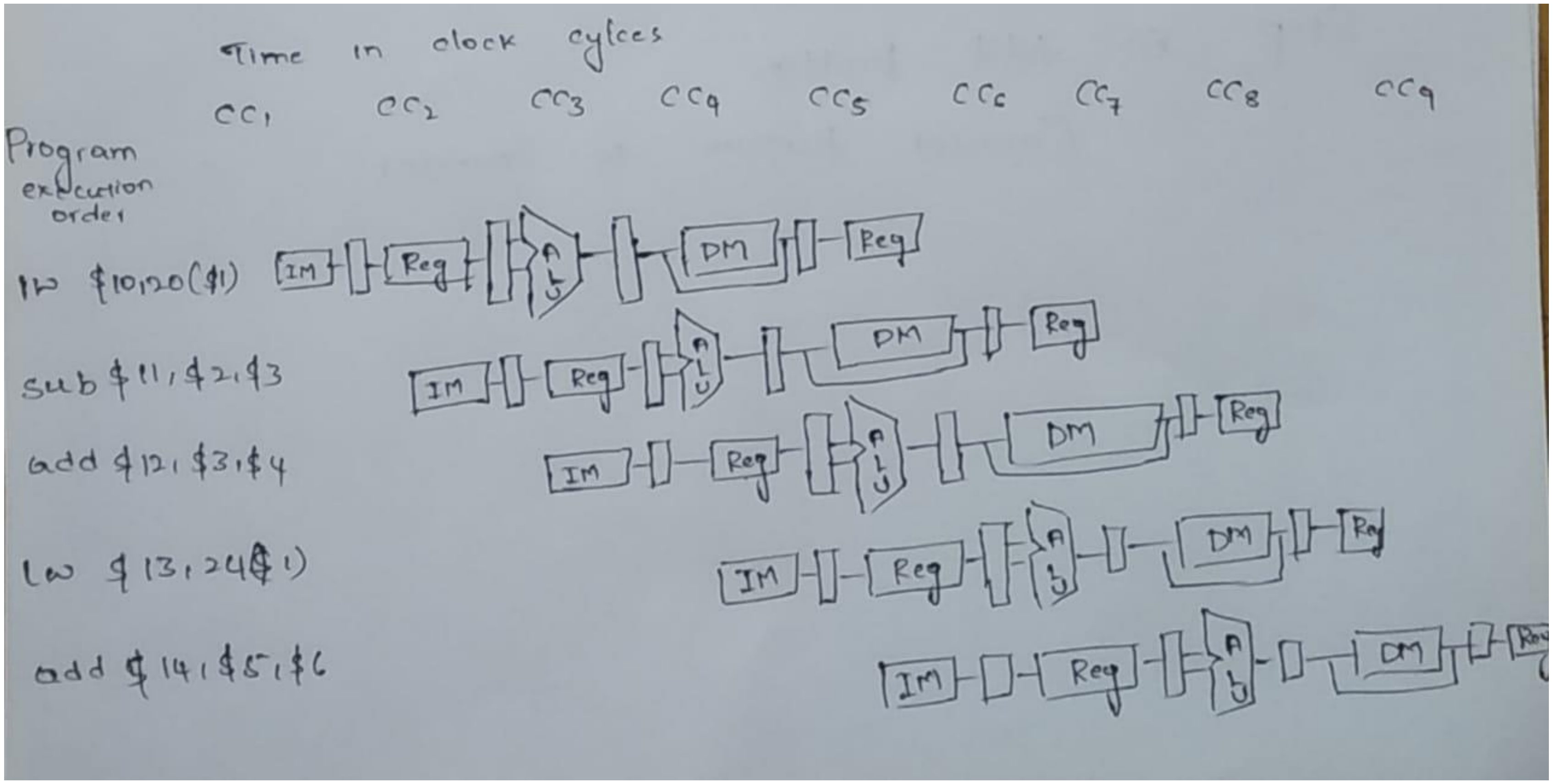
w $10, 20($1)

ub $11, $2, $3

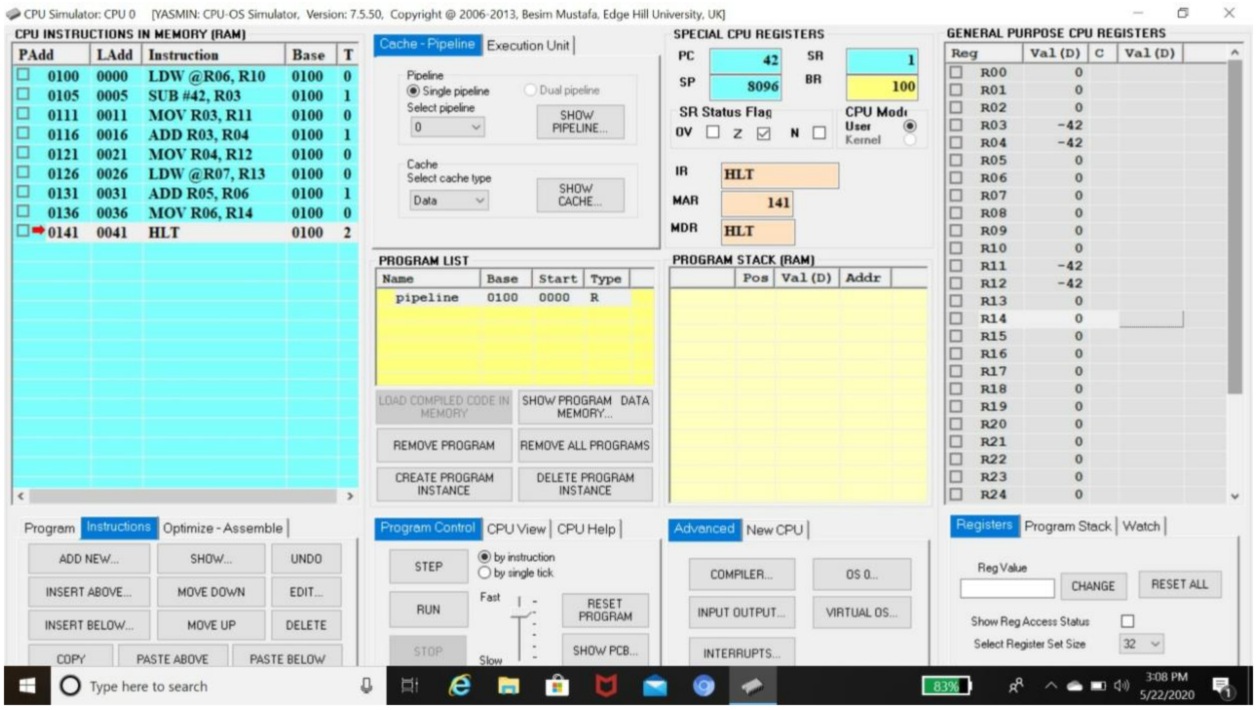
dd $12, $3, $4

w $13, 24($1)

dd $14, $5, $6


Time in clock cylces
CC1
CC2
CC3
CC4
CC5
CC6
CC7
CC8
CC9
Program execution order
lw $10,20($1)
sub $11,$2,$3
add $12,$3,$4
lw $13,24($1)
add $14,$5,$6
IM
Reg
ALU
DM
Reg

SNAPSHOTS



CPU Simulator: CPU 0 [YASMIN: CPU-OS Simulator, Version: 7.5.50, Copyright @ 2006-2013, Besim Mustafa, Edge Hill University, UK]

CPU INSTRUCTIONS IN MEMORY (RAM)

PAdd	LAdd	Instruction	Base	T
0100	0000	LDW @R06, R10	0100	0
0105	0005	SUB #42, R03	0100	1
0111	0011	MOV R03, R11	0100	0
0116	0016	ADD R03, R04	0100	1
0121	0021	MOV R04, R12	0100	0
0126	0026	LDW @R07, R13	0100	0
0131	0031	ADD R05, R06	0100	1
0136	0036	MOV R06, R14	0100	0
0141	0041	HLT	0100	2

Pipeline: [✓ Single pipeline] Dual pipeline; Select pipeline: 0
Cache: Select cache type: Data

PROGRAM LIST

Name	Base	Start	Type
pipeline	0100	0000	R

SPECIAL CPU REGISTERS

PC 42 SR 1
SP 8096 BR 100
SR Status Flag: OV ☐ Z ☑ N ☐
CPU Mode: [✓ User] Kernel
IR HLT
MAR 141
MDR HLT

PROGRAM STACK (RAM)

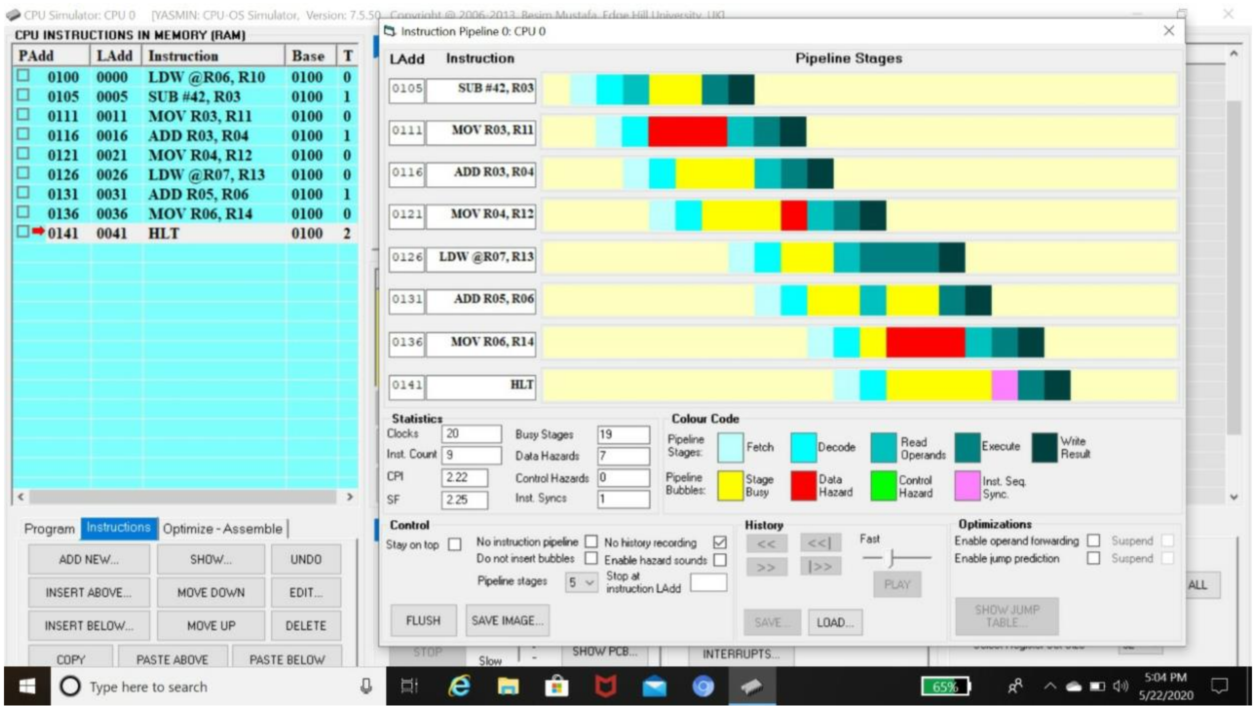
Pos	Val (D)	Addr

GENERAL PURPOSE CPU REGISTERS

Reg	Val (D)	C	Val (D)
R00	0		
R01	0		
R02	0		
R03	-42		
R04	-42		
R05	0		
R06	0		
R07	0		
R08	0		
R09	0		
R10	0		
R11	-42		
R12	-42		
R13	0		
R14	0		
R15	0		
R16	0		
R17	0		
R18	0		
R19	0		
R20	0		
R21	0		
R22	0		
R23	0		
R24	0		

Program Control: [✓ by instruction] by single tick
Show Reg Access Status ☐
Select Register Set Size: 32

3:08 PM 5/22/2020



Instruction Pipeline 0: CPU 0

LAdd	Instruction
0105	SUB #42, R03
0111	MOV R03, R11
0116	ADD R03, R04
0121	MOV R04, R12
0126	LDW @R07, R13
0131	ADD R05, R06
0136	MOV R06, R14
0141	HLT

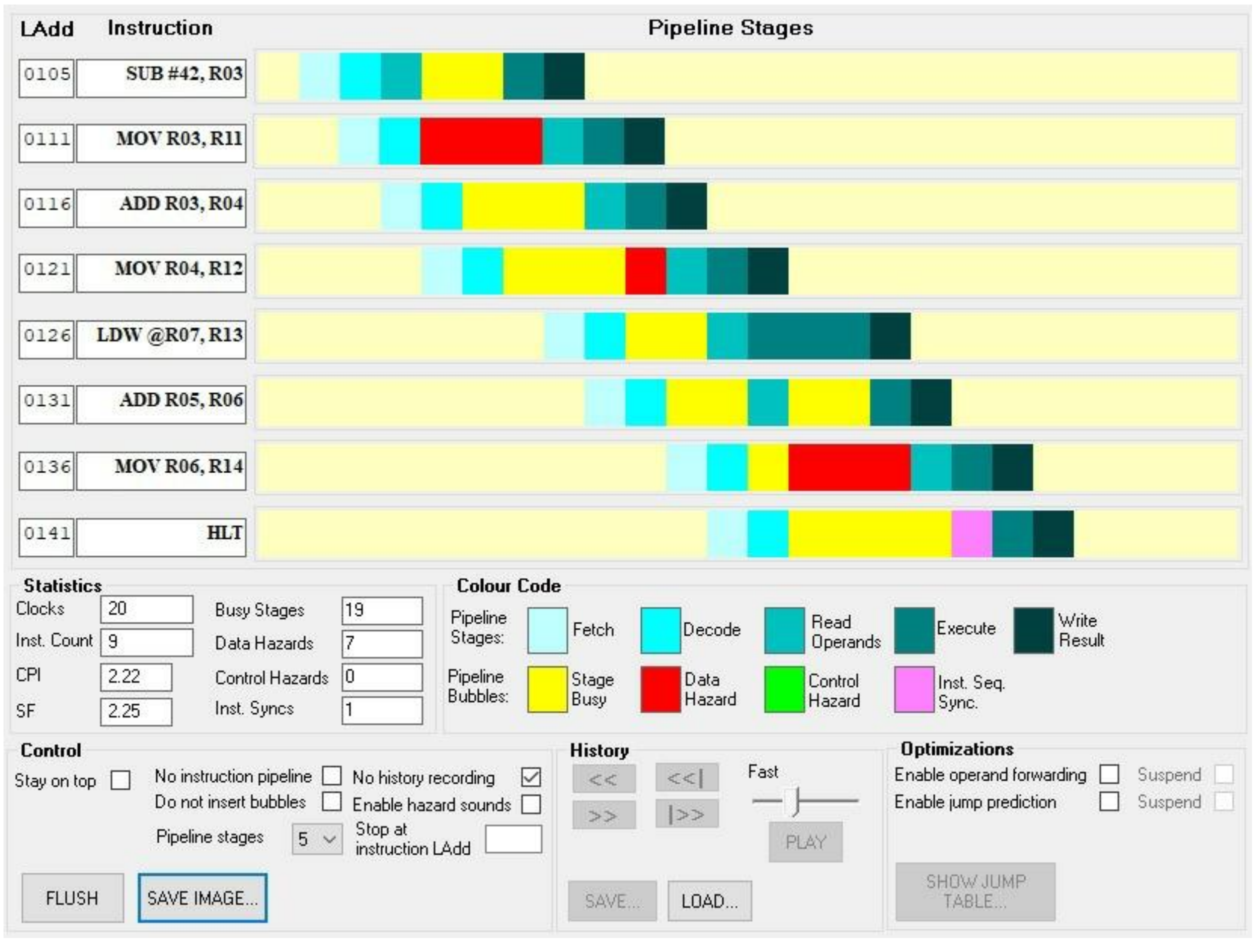
Statistics

Clocks	20	Busy Stages	19
Inst. Count	9	Data Hazards	7
CPI	2.22	Control Hazards	0
SF	2.25	Inst. Syncs	1

Colour Code
Pipeline Stages: Fetch, Decode, Read Operands, Execute, Write Result
Pipeline Bubbles: Stage Busy, Data Hazard, Control Hazard, Inst. Seq. Sync.

Control: Stay on top ☐; No instruction pipeline ☐; Do not insert bubbles ☐; No history recording ☑; Enable hazard sounds ☐; Pipeline stages 5; Stop at instruction LAdd
Optimizations: Enable operand forwarding ☐ Suspend; Enable jump prediction ☐ Suspend

5:04 PM 5/22/2020



LAdd	Instruction
0105	SUB #42, R03
0111	MOV R03, R11
0116	ADD R03, R04
0121	MOV R04, R12
0126	LDW @R07, R13
0131	ADD R05, R06
0136	MOV R06, R14
0141	HLT

Statistics

Clocks	20	Busy Stages	19
Inst. Count	9	Data Hazards	7
CPI	2.22	Control Hazards	0
SF	2.25	Inst. Syncs	1

Colour Code

Pipeline Stages: Fetch, Decode, Read Operands, Execute, Write Result

Pipeline Bubbles: 🟨 Stage Busy, 🟥 Data Hazard, 🟩 Control Hazard, 🟪 Inst. Seq. Sync.

Control

Stay on top ☐ No instruction pipeline ☐ No history recording ☑

Do not insert bubbles ☐ Enable hazard sounds ☐

Pipeline stages 5 Stop at instruction LAdd

Optimizations

Enable operand forwarding ☐ Suspend ☐

Enable jump prediction ☐ Suspend ☐