Part 1: Research Paper Exploration

Summary:

Key Objectives

This paper aims to review the fundamentals of computer architecture simulation techniques, comparing different ones to each other. The main objective is to help computer architects pick the right kind of simulation for their field by classifying different types based on their features, capabilities, and input/output types.

Methodologies

Classification
Simulators are categorized into different groups based on multiple different metrics:

- Simulation Detail:
  This refers to how much information the simulation in question provides and in how much detail and precision.

- Scope:
  This refers to the simulation's versatility; this part takes into account multiple things such as level of detail, input/output type and variety, as well as its overall information handling capability (how much can it handle?).

- Input Type:
  This refers to what kind of input the simulation can take in. Some simulations can only take in numerical or character values or run a specific, predetermined set of instructions while others can run basic code on them.

Comparison
Detailed comparisons of six contemporary x86 simulators (gem5, MARSSx86, Multi2Sim, PTLsim, Sniper, ZSim) are made based on flexibility, micro-architectural details, and experimental error.

Validation
This paper measures the simulations' level of error and compares them against real hardware to test their accuracy to physical components.

Findings

Simulator Categories

Simulators are grouped into 3 different categories:

- Functional:
  This category focuses on real world fidelity and accuracy. The simulators are closely modeled after real world architecture, sacrificing certain extra information (“microarchitectural parameters”), in favor of mimicking the simulated hardware as closely as possible.

- Timing:
  These simulators focus more on giving information about a certain process's timing and performance, such as the length of an instruction cycle for a CPU. There are 3 sub-categories for timing simulators: cycle-level, event-driven and interval.

- Integrated:
  Simply put, these simulators have both functional and timing capabilities, allowing for a more detailed simulation overall.

Performance

The paper highlights the trade-offs between simulation speed and accuracy, with functional simulators being faster but less detailed than timing simulators.

Validation Challenges

The paper highlights the challenges in validating simulation results, emphasizing the need for accurate documentation and comparison with real hardware.

Experimental Results

The evaluated simulators displayed different levels of accuracy, each with different strengths and weaknesses.

Critical Analysis:

Advantages

- **Improved Accuracy in Simulations**:
  Certain simulations such as Sniper and ZSim provide higher accuracy results than the other tested simulators. This high accuracy is essential for reliable results, which is highly important for research and development in computer architecture.

- **Flexibility and Customization**:
  Tools like gem5 and MARSSx86 offer more flexibility and customization for modeling new micro-architectural features. This allows researchers to test new design ideas and study the performance of specific micro-architectural blocks, making these tools highly valuable for academic and industrial research.

- **Enhanced Performance for Specific Architectures**:
  Simulators such as Sniper and ZSim are optimized for many-core x86 architectures, providing faster simulation speeds and better accuracy for these systems. This optimization is particularly advantageous for studies focused on many-core processor architectures, enabling more efficient and effective research.

Potential Limitations

- **Inflexibility in Modeling New Features**:
  While Sniper and ZSim offer high accuracy, they are less flexible for modeling new micro-architectural features compared to other simulators like gem5 and MARSSx86. This limitation can hinder research that requires extensive customization and new feature integration.

- **Validation and Accuracy Issues**:
  The accuracy of simulators can vary significantly, leading to potential inaccuracies in simulation results. Unvalidated or improperly calibrated simulators can produce results that diverge considerably from real hardware performance. This issue underscores the importance of rigorous validation and calibration processes to ensure reliable simulation outcomes.

Application:

- **Category Benefits:**
  The different simulation categories give a new view on how stimulators actually function, as well as their distinct differences. By making use of these categories as well as each of their accompanying factors (detail, scope, and input type), I can more easily determine which simulation type best suits the current task at hand, and proceed accordingly.

- **Trade-Off Knowledge:**
  Reading this paper better helped understand the trade-offs between speed and accuracy in functional and timing simulators. Depending on the situation, whether it's basic research or testing for a proof, this information could prove useful especially when aiming for reliable results.

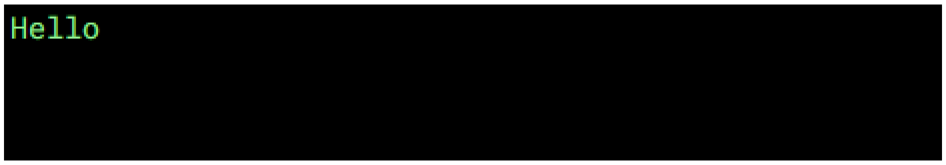
- **Documentation and Explanation:**
  The paper's structure itself can also be applied to course activities, as its detailed and clean structure as well as its overall well segments categories allow for readers to better understand what's being explained. Based on my previous experience in writing explanations for this course's activities, I could seriously benefit from such a structure.

Part 2: Evaluating emulsiV

Overview:

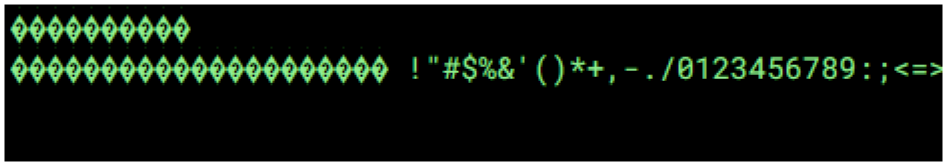
emulsiV is a CPU simulator that showcases the steps that a processor takes to accomplish a given task. Using premade programs provided by the website or by letting the user alter certain instructions themselves, emulsiV gives a visual representation of every single step, and further breaks it down by showing every single bit that travels through the different components of the processor for every step. The website features a bunch of different presets such as “Hello” which prints the word “Hello” in the textbox, “ASCII” which displays every ASCII character in order, “Bitmap” which draws an image on a bitmap, and more. It also has the ability to let the user input their own instructions into the memory, creating custom programs. The website also has a manual that teaches everything the user needs to know about what everything (instruction abbreviations, bit values, etc.) means. As an added bonus, the user can also choose from a variety of display options for the instructions themselves. Finally, the user can toggle between having no animation, or having each step animated, showing the trajectory of each bit and modifiable speeds for easier tracking.

“Hello” Program:



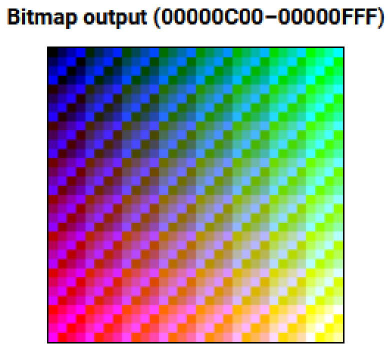
```
Hello
```

“ASCII” Program:



```
����������
����������������������������� !"#$%&'()*+,-./0123456789:;<=>
```

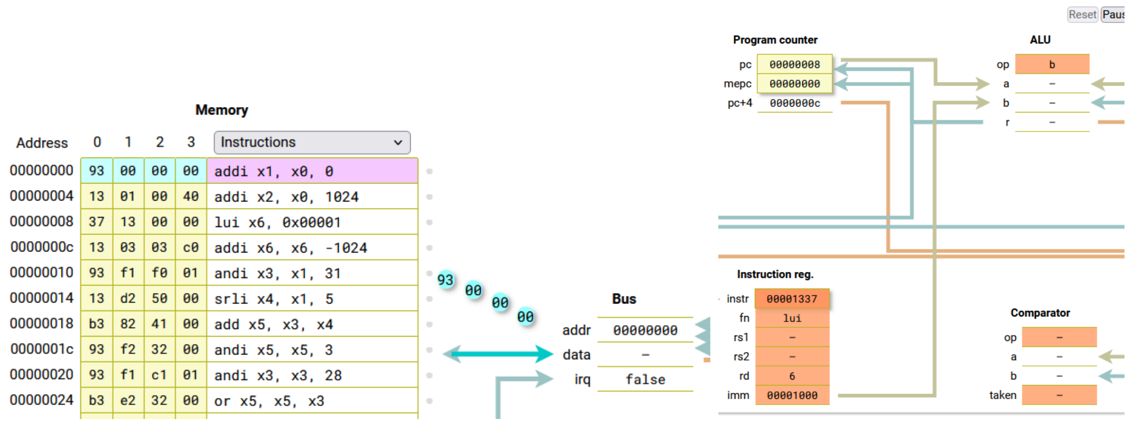
“Bitmap” Program:



Advantages and Disadvantages:

Advantages

- **User Friendly and Simple to Understand:**
  emulsiV makes learning how a CPU works very easy. The animation option makes everything simple to keep track of and understand by color coding all moving parts. The parts in blue are either a start point, an end point, or a moving piece of information, making it clear from where the bits start and where they're going. The parts in orange mark a change of information, signifying that a certain cell has been modified by either accepting new data, or by having an operation performed on it. These two colors make each sub-step very simple to follow and differentiate between.



Address	0	1	2	3	Instructions
00000000	93	00	00	00	addi x1, x0, 0
00000004	13	01	00	40	addi x2, x0, 1024
00000008	37	13	00	00	lui x6, 0x00001
0000000c	13	03	03	c0	addi x6, x6, -1024
00000010	93	f1	f0	01	andi x3, x1, 31
00000014	13	d2	50	00	srli x4, x1, 5
00000018	b3	82	41	00	add x5, x3, x4
0000001c	93	f2	32	00	andi x5, x5, 3
00000020	93	f1	c1	01	andi x3, x3, 28
00000024	b3	e2	32	00	or x5, x5, x3

Bus

addr	00000000
data	–
irq	false

Program counter

pc	00000008
mepc	00000000
pc+4	0000000c

ALU

op	b
a	–
b	–
r	–

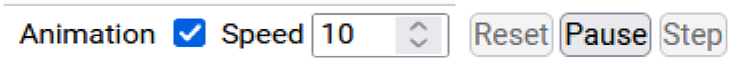
Instruction reg.

instr	00001337
fn	lui
rs1	–
rs2	–
rd	6
imm	00001000

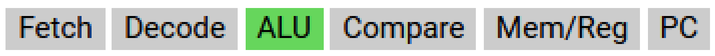
Comparator

op	–
a	–
b	–
taken	–

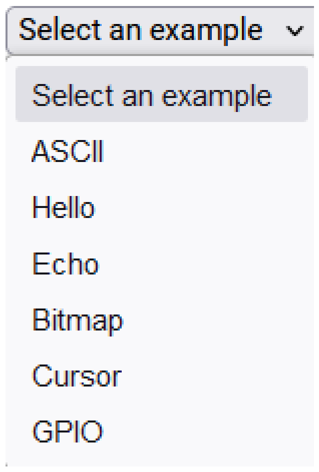
On top of that, there's even the option to change the information's movement speed, slowing it down in order to better understand each individual sub-step, and speeding it back up if the process becomes repetitive. It also has the option to pause the process mid step, as well as to run the progress step by step, stopping it automatically when an instruction has been completed, allowing for users to keep track of when one step ends and the other one begins.


Animation ✓ Speed 10

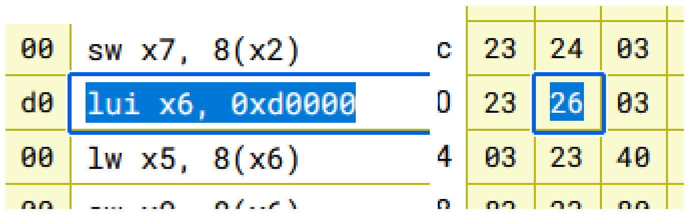
Finally, emulsiV lights up which part of the process the current instruction is in, further dividing a sub-step into even more information.



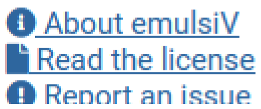
- **Beginner and Option Friendly:**
  Aside from being easy to track and understand, emulsiV comes with multiple different program presets, allowing for users who aren't too familiar with the inner workings of a CPU to use the website without needing any prior experience in the field.



  On the other hand, for those who have a bit more experience, notably in Assembly, or for those who are simply curious, there's also the option to write custom instructions and data. A detailed manual explaining the different commands and data sets is also provided in the "About EmulsiV" hyperlink on the bottom right of the site.

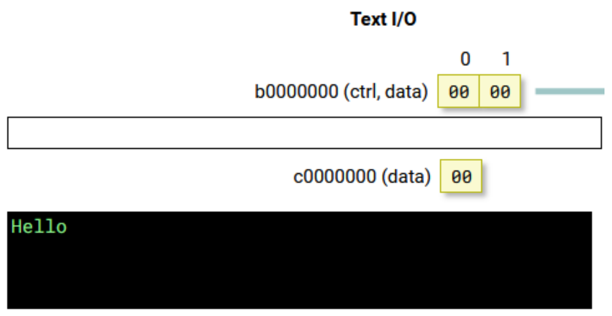


```
00  sw x7, 8(x2)       c  23 24 03
d0  lui x6, 0xd0000    0  23 26 03
00  lw x5, 8(x6)       4  03 23 40
```



- **Highly Versatile:**
  emulsiV supports a wide variety of processes, including characters and numbers, bitmap displays and buttons, diving quite deep into the inner workings of those separate mechanics.

  Test Display:



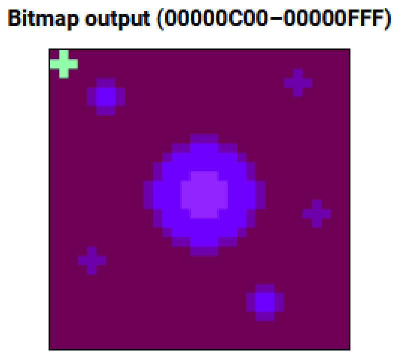
Text I/O

	0	1
b0000000 (ctrl, data)	00	00

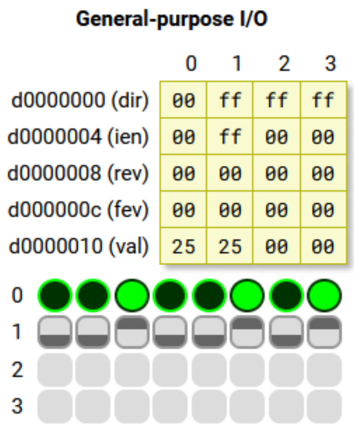
c0000000 (data)	00

```
Hello
```

  Bitmap Display:



  Button Display:



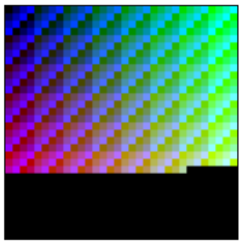
General-purpose I/O

	0	1	2	3
d0000000 (dir)	00	ff	ff	ff
d0000004 (ien)	00	ff	00	00
d0000008 (rev)	00	00	00	00
d000000c (fev)	00	00	00	00
d0000010 (val)	25	25	00	00

Disadvantages

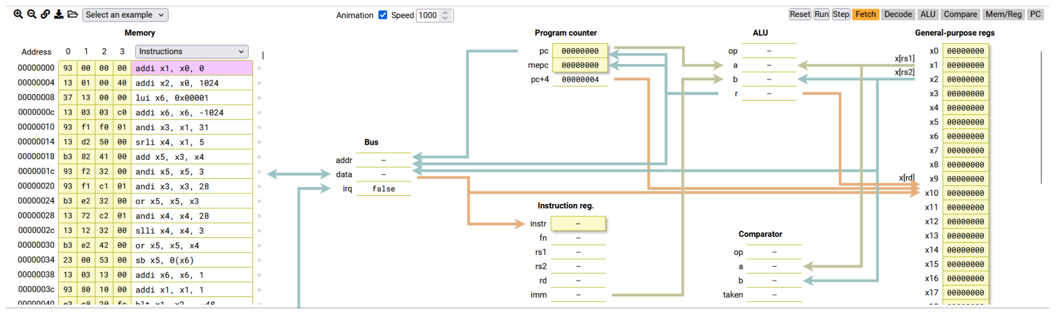
- **Animation Speed Limitations:**
  While the animation speed can be increased to a pretty high number (over 1000), there seems to be an underlying issue for longer processes, especially custom ones. If we take the bitmap process as an example, each pixel takes around 5 seconds to appear when animations are toggled on (and the animation speed is at max). Turning off animations would make this entire process instant, but prevent a step by step learning chance. While this specific process is repetitive, and doesn't need any animations to understand past the first few pixels, an issue arises if users input long custom processes, in which there would be no way to easily or quickly reach a midpoint in the process for examining.

Bitmap output (00000C00–00000FFF)



- **Purely Theoretical Simulation:**
  Simply put, while this simulation is quite good at teaching the basics of how a CPU works, its structure only covers the bare minimum and cannot be used to accurately represent a real world processor. To add, the components are displayed in a way that is better suited for visual learning, in turn making it less accurate compared to real hardware.

  emulsiV: Structurally different than real hardware



Speed 1000

Memory

Address	0	1	2	3	Instructions
00000000	93	00	00	00	addi x1, x0, 0
00000004	13	01	00	40	addi x2, x0, 1024
00000008	37	13	00	00	lui x6, 0x00001
0000000c	13	03	03	c0	addi x6, x6, -1024
00000010	93	f1	f0	01	andi x3, x1, 31
00000014	13	d2	50	00	srli x4, x1, 5
00000018	b3	82	41	00	add x5, x3, x4
0000001c	93	f2	32	00	andi x5, x5, 3
00000020	93	f1	c1	01	andi x3, x3, 28
00000024	b3	e2	32	00	or x5, x5, x3
00000028	13	72	c2	01	andi x4, x4, 28
0000002c	13	12	32	00	slli x4, x4, 3
00000030	b3	e2	42	00	or x5, x5, x4
00000034	23	00	53	00	sb x5, 0(x6)
00000038	13	03	13	00	addi x6, x6, 1
0000003c	93	80	10	00	addi x1, x1, 1

Program counter

pc	00000000
mepc	00000000
pc+4	00000004

ALU

op	–
a	–
b	–
r	–

Bus

addr	–
data	–
irq	false

Instruction reg.

instr	–
fn	–
rs1	–
rs2	–
rd	–
imm	–

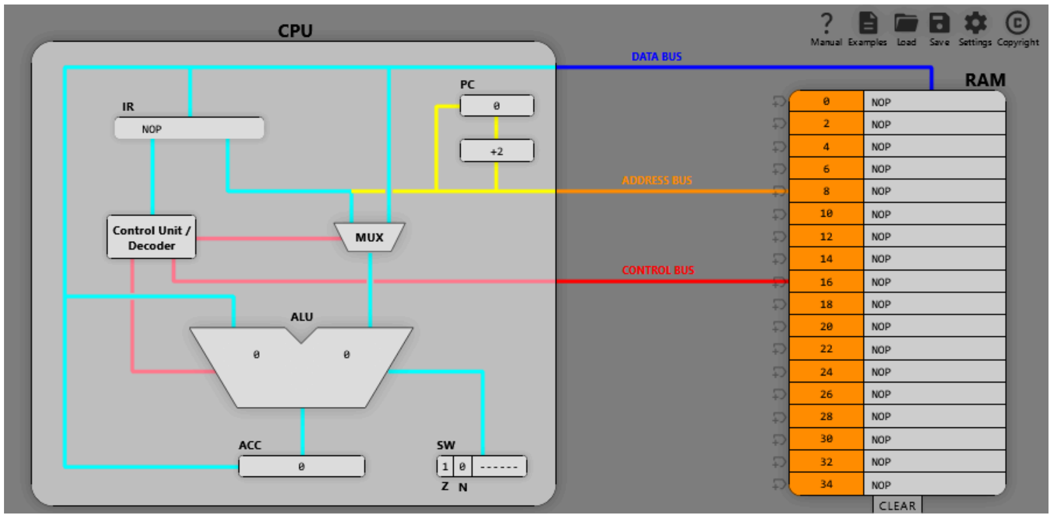
Comparator

op	–
a	–
b	–
taken	–

General-purpose regs

x0	00000000
x1	00000000
x2	00000000
x3	00000000
x4	00000000
x5	00000000
x6	00000000
x7	00000000
x8	00000000
x9	00000000
x10	00000000
x11	00000000
x12	00000000
x13	00000000
x14	00000000
x15	00000000
x16	00000000
x17	00000000

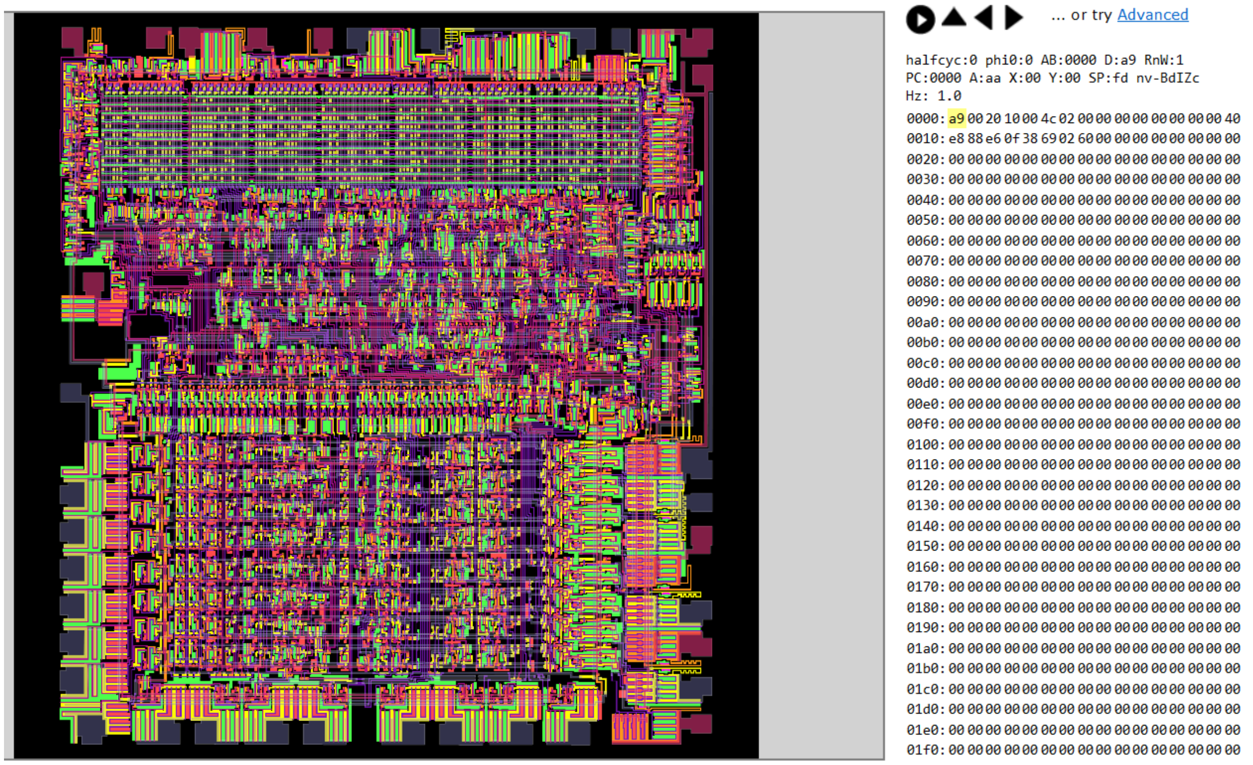
CPU Visual Simulator (https://cpuvisualsimulator.github.io/): Visually more accurate



```
PC: 0
+2
IR: NOP
ALU: 0   0
ACC: 0
SW: 1 | 0 | ------
```

0	NOP
2	NOP
4	NOP
6	NOP
8	NOP
10	NOP
12	NOP
14	NOP
16	NOP
18	NOP
20	NOP
22	NOP
24	NOP
26	NOP
28	NOP
30	NOP
32	NOP
34	NOP

Visual 6502 (http://www.visual6502.org/JSSim/index.html): Incredibly accurate



```
halfcyc:0 phi0:0 AB:0000 D:a9 RnW:1
PC:0000 A:aa X:00 Y:00 SP:fd nv-BdIZc
Hz: 1.0
0000: a9 00 20 10 00 4c 02 00 00 00 00 00 00 00 00 40
0010: e8 88 e6 0f 38 69 02 60 00 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

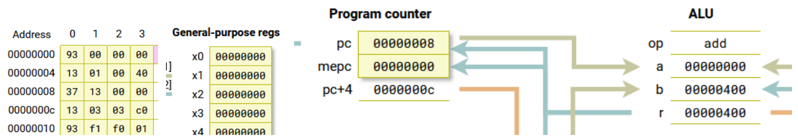
Comparison with Traditional Tools:

Comparison Between emulsiV (https://eseo-tech.github.io/emulsiV/) and CPU Visual Simulator (https://cpuvisualsimulator.github.io/):

- **Performance:**
  While CPU Visual Simulator (referred as CVS from now on) is meant to be a more simple simulator compared to emulsiV, its overall animated and non-animated speeds are both considerably slower than that of emulsiV, making the overall waiting process more tedious. However, as shown above, CVS's structure is of a higher fidelity than that of emulsiV, making it slightly more accurate for real hardware cases. Also, based simply off of observation, CVS appears to be a purely functional simulator while emulsiV looks like an integrated one, having both functional and timing components working together (which is why emulsiV has more intermediate steps).

- **Ease of Use:**
  Both simulators have a pretty in depth manual explaining what each instruction does, which is a plus on both ends. However, emulsiV has a more visual and interactive system that not only displays what information is stored in the memory and the registers, but also shows each data set's trip through the different components of the processor (as opposed to the very few values that change in the CVS).

  emulsiV's high information levels, displayed in hexadecimal 8 bit segments



Address	0	1	2	3
00000000	93	00	00	00
00000004	13	01	00	40
00000008	37	13	00	00
0000000c	13	03	03	c0
00000010	93	f1	f0	01

General-purpose regs

x0	00000000
x1	00000000
x2	00000000
x3	00000000
x4	00000000

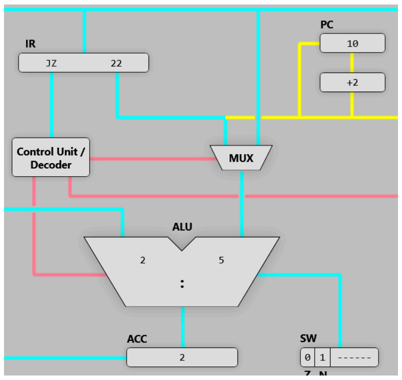
Program counter

pc	00000008
mepc	00000000
pc+4	0000000c

ALU

op	add
a	00000000
b	00000400
r	00000400

  CVS's low information display, shown only in minimal decimal numbers



- **Application Areas:**
  As mentioned above, CVS has a higher accuracy to real hardware, meaning that it's slightly better to use that one for hardware based applications. emulsiV however, has many more features like the aforementioned process types (characters, bitmaps, buttons) as well as the numerous display types and indicators, making it way better for pure theory based applications, as well as more software inclined ones.

Practical Application:

Since emulsiV supports button commands that users can interact with, a good research scenario could revolve around that concept. A hypothetical research topic could be analyzing real time user to computer interactions with the goal of finding the fastest way (cycle wise) to receive an input from the user (in this case a button press on the emulsiV interface) and then display the corresponding output. Since emulsiV has a high information display level as well as a bunch of customization options including programmable instructions, the entire project could in theory be conducted and optimized in this single website. To expand on this, the project could also aim to take the shortest of every metric (cycle, time, etc.), average them out, and find the most optimal process for this simple interaction.

The only references used was the required research paper as well as the simulators seen in class:

A. Akram and L. Sawalha, "A Survey of Computer Architecture Simulation Techniques and Tools," in IEEE Access, vol. 7, pp. 78120-78145, 2019, doi: 10.1109/ACCESS.2019.2917698. https://ieeexplore.ieee.org/document/8718630

Simulations:
https://eseo-tech.github.io/emulsiV/
https://cpuvisualsimulator.github.io/
http://www.visual6502.org/JSSim/index.html