Effective Approach for Teaching Computer System Architecture

K. M. Hasam¹, Akhlaq A Khan², M. Saleem³, M Riaz Moghal⁴, Yasir Mahmood⁴, M. Adnan⁵, Tanveer Akhtar⁶

¹,³ Department of Electrical Engineering, University of Engineering and Technology, Lahore.
² Lecturer, Department of Computer Science and Information Technology, Mirpur (AJK)-10250
⁴ Chairman, Department of Computer Science and Information Technology, Mirpur (AJK)-10250
² University College of Engineering and Technology, Mirpur (AJK)-10250, Pakistan
⁵,⁶,⁷ Graduate/Undergraduate Student, Department of Computer Sciences and Information Technology, Mirpur (AJK)-10250, Pakistan

Abstract: - The present Basic Computer is a concept that demonstrates the least features that a digital computer must exhibit. To make itself to be called a general purpose computer; the Basic Computer is able to execute a stored program in automatic fashion. It has the ability to fetch, decode and execute packed binary instructions. Its architectural design promises the minimum functionality through the ability of sequential execution, conditional and unconditional jumps, loops, interrupt facility for efficient I/O performance and mechanism to introduce sub-routines into the programs written for the Basic Computer. Thus introducing almost all conventional programming constructs. The design of Basic Computer defines a limited instruction set in three categories: 1) Memory Reference Instructions (MRI) that allow processes on memory operands, 2) Register Reference Instructions whose domain of action is register set, 3) I/O instructions that provide basic Input/Output facility. A symbolic assembly language offers a bit higher level of interaction, thus enabling Basic Computer demonstrate the idea of language development and language processing. The complete design consists of i) A Common-Bus Architecture, ii) A Register Set, iii) A flag Set, iv) A memory Unit, v) A Hardwired Control Unit, vi) An Arithmetic and Logic Unit.

The present product itself is a learning tool (that runs on Microsoft Windows® on Intel® Architecture). It consists of following components: the IDE, the VM, the Program Tracer, the CPU Tracer. It is equally useful for both student and teacher. The teacher will find it to be an excellent instruction instrument. He/she is able to communicate the complex ideas of computer design because of the visual demonstration offered by the product. For student, the application serves as an electronic teacher, as it offers so many learning features.

Key-Words: - Hardware Design, Computer Architecture, Assembly Language, IDE, Hardwired Control Unit.

1 Introduction

Undergraduate students in computer architecture teaching and learning of technical courses have ever been difficult and challenging tasks due to the complexity of the subjects. Particularly in the subjects that are aimed at understanding some physical model, and where the functional speed of the model is many times faster than the human perception speed, the difficulty in communicating the details of that model is likely be more problematic. Many physical models exhibit functional parallelism; that is, at certain point on the time-line, so many events take place simultaneously. Although it varies individual to individual, yet an average individual rarely demonstrates his/her ability to perceive functional parallelism.

During the study of computer science courses, particularly those related to the computer system architecture and organization, we observed difficulty in communication and perception of functional concepts of computer. Modern computers perform so many tasks (micro operations) at a single clock pulse. When these tasks are interdependent, or otherwise are the steps of some sequence, their verbal interpretation does not elegantly fulfill the teaching purpose. The idea of Computer Aided Instruction (CAI[1]) is not new, but CAI products are rarely seen that can be used to demonstrate architecture, organization, structure and function of digital computers. One existing system is Emu8086[2], but it traces program instruction by instruction. We gained the idea that an experimental project should be undertaken keeping in view the following points: (a) Initially less complex computer system should be simulated to facilitate the product development process as well as its presentation in Computer Architecture class. (b) The simulated model should demonstrate all those (least) features that should qualify it to be the general purpose digital computer. (c) The design details should be visualized. (d) The functional details should be visualized.

Proceeding with this theme, we developed a product named “Basic Computer Simulator (BCS)” which is the simulation of a complete computer design. The concept of Basic Computer has
originally been presented by M. Morris Mono in his book Computer System Architecture [3]. We selected this model for simulation. The Basic Computer consists of a memory unit with 4096 words of 16 bits each, eight registers: AR, PC, DR, AC, IR, TR, OUTR and INPR, one sequence counter SC, seven flip-flops: I, S, E, R, IEN, FG1, and FGO, two decoders: a 3\(\times\)8 operation decoder and a 4\(\times\)16 timing decoder, a 16-bit common bus, hardwired control units and an adder and logic circuit connected to the input of AC. The simulation consists of an Integrated Development Environment (IDE) that further consists of a code editor and a built-in assembler, a virtual machine, a program tracer and a CPU tracer. A built-in help system accompanies the product which provides content based help, keyword search, context sensitive help and “What is this?” help.

To develop product, we required a big knowledgebase in following areas: Digital Logic and Design[4], Modern Computer Architecture[5], Automata Theory[6], Compiler Construction[7], Programming Visual C[8], Programming Visual Basic[9], Win32 Standard DLL technology[10], Programming Assembly Language[11] and interface designing. We studied the material on these subjects in books and on the World Wide Web.

After completing the product, we tested it in the classroom and observed its effects on students’ learning process. We gathered that it enhanced the learning process more than our expectations and transmitted every detail of not only the model the simulation is of, but also the general computer architecture and organization concepts. We are of the opinion that the benefit of such products is two fold. On one side it enhances the intellect of the developers, and on the other side it enhances the class-work efficiency from both students and teachers perspective.

2 Basic Computer Simulator

2.1 Software Architecture
Here we present an overall view of software architecture. The complete system consists of following components:


a) Basic Computer Virtual Machine: This is a logical component. When this component is loaded, the physical system starts behaving like Basic Computer. The machine code for Basic Computer runs on this virtual machine. The virtual machine itself runs on physical platform.

b) Program Tracer: This is an interface component that visualizes line-by-line program execution. This runs itself on physical platform but it executes Basic Computer programs on virtual machine.

c) CPU Tracer: This is another interface component that visualizes program execution on clock based tracing. This also runs on physical platform but executes Basic Computer programs on virtual machine.

Figure 1 depicts the accommodation of these components within the system. At the lowest level is the physical platform: Microsoft Windows® on Intel® Architecture. The Integrated Development Environment runs directly on the physical platform. So these components have nothing to do with the Basic Computer Virtual Machine. The user types code in code editor and then invokes the built-in assembler to assemble the code into the target machine language. The assembler also generates some listing information in addition to the machine code. This listing information is used by the Program Tracer and the CPU tracer to visualize execution process. The Program Tracer and the CPU Tracer run programs on Basic Computer Virtual Machine which, in turn, runs on physical platform.

The Virtual Machine, the Program Tracer and the CPU tracer are the most complex components of the system.

2.2 Common Bus System
The basic computer has eight registers, a memory unit, and a control unit. Paths must be provided to transfer information from one register to another and
between memory and registers. The number of wires will be excessive if connections are made between the outputs of each register and the inputs of the other registers. A more efficient scheme for transferring information in a system with many registers is to use a common bus. A bus system can be constructed using a combination of multiplexers. The connections of the registers and memory of the basic computer to a common bus system is shown in Figure-2.

The outputs of six registers and memory are connected to the common bus. The specific output that is selected for the bus lines at any given time is determined from the binary value of the selection variables $S_2, S_1$, and $S_0$. The number along each output shows the decimal equivalent of the required binary selection. For example, the number along the output of $DR$ is 3. The 16-bit outputs of $DR$ are placed on the bus lines when $S_2S_1S_0 = 011$ since this is the binary value of decimal 3. The lines from the common bus are connected to the inputs of each register and the data inputs of the memory. The particular register whose $LD$ (load) input is enabled receives the data from the bus during the next clock pulse transition. The memory receives the contents of the bus when its write input is activated. The memory places its 16-bit output onto the bus when the read input is activated and $S_2S_1S_0 = 111$.

![Figure 2: Basic computer registers connected to common bus](image)

Four registers, $DR, AC, IR,$ and $TR$, have 16 bits each. Two registers, $AR$ and $PC$, have 12 bits each since they hold a memory address. When the contents of $AR$ or $PC$ are applied to the 16-bit common bus, the four most significant bits are set to 0's. When $AR$ or $PC$ receives information from the bus, only the 12 least significant bits are transferred into the register. The input register $INPR$ and the output register $OUTR$ have 8 bits each and communicate with the eight least significant bits in the bus. $INPR$ is connected to provide information to the bus but $OUTR$ can only receive information from the bus. This is because $INPR$ receives a character from an input device which is then transferred to $AC$. $OUTR$ receives a character from $AC$ and delivers it to an output device. There is no transfer from $OUTR$ to any of the other registers.

The 16 lines of the common bus receive information from six registers and the memory unit. The bus lines are connected to the inputs of six registers and the memory. Five registers have three control inputs: $LD$ (load), $INR$ (increment), and $CLR$ (clear). This type of register is equivalent to a binary counter with parallel load and synchronous clear.

The increment operation is achieved by enabling the count input of the counter. Two registers have only a $LD$ input. The input data and output data of the memory are connected to the common bus, but the memory address is connected to $AR$. Therefore, $AR$ must always be used to specify a memory address. By using a single register for the address, we eliminate the need for an address bus that would have been needed otherwise. The content of any register can be specified for the memory data input during a write operation. Similarly, any register can receive the data from memory after a read operation except $AC$. The 16 inputs of $AC$ come from an adder and logic circuit. This circuit has three sets of inputs. One set of 16-bit inputs come from the outputs of $AC$. They are used to implement register microoperations such as complement $AC$ and shift $AC$. Another set of 16-bit inputs come from the data register $DR$. The inputs from $DR$ and $AC$ are used for arithmetic and logic microoperations, such as ADD $DR$ to $AC$ or AND $DR$ to $AC$. The result of an addition is transferred to $AC$ and the end carry-out of the addition is transferred to flip-flop $E$ (extended $AC$ bit). A third set of 8-bit inputs comes from the input register $INPR$.

Note that the content of any register can be applied onto the bus and an operation can be performed in the adder and logic circuit during the same clock cycle. The clock transition at the end of the cycle transfers the content of the bus into the designated destination register and the output of the adder and logic circuit into $AC$. For example, the two microoperations $DR ← AC$ and $AC ← DR$ can be executed at the same time. This can be done by placing the content of $AC$ on the bus (with $S_2S_1S_0 = 100$), enabling the $LD$ (load) input of $DR$, transferring the content of $DR$ Through the adder and logic circuit into $AC$, and enabling the $LD$ (load) input of $AC$, all during the same clock cycle. The two transfers occur upon the arrival of the clock pulse transition at the end of the clock cycle.

### 2.3 The Instruction Set

The instruction set of Basic Computer consists of a
total of 25 instructions in three categories. There are seven memory-reference instructions (MRI): AND, ADD, LDA, STA, BUN BSA and ISZ. MRIs are those instructions which require a memory operand. Other 18 instructions, which we call non-MRIs, are classified further as Register Reference Instruction and I/O Instructions. Register Reference Instruction are 12 in number and operate on CPU register AC. The I/O instructions are aimed at input/output and are 6 in number. The computer has two addressing modes: direct addressing and indirect addressing. Therefore every MRI can appear in any of two possible forms: with direct memory operand or with indirect memory operand. In direct MRIs, the address part of instruction contains the effective address (the memory address where operand value can be found), while the address part of indirect MRIs contains the address of effective address. The instruction set consists of minimum number of instruction required for a general purpose computer to be functional. The ability of sequential execution, loops and jumps is available and special instructions are dedicated to these. The ALU is simple one and can perform ADD, AND, and NOT operation. The AC can be incremented or cleared desirably. All other arithmetic and logical operations can be obtained through programming.

Table-1: Complete instruction set of Basic Computer

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Hexadecimal code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRE</td>
<td>I = 0</td>
<td>I = 1</td>
</tr>
<tr>
<td>AND 0xxx</td>
<td>8xxx</td>
<td>AND memory word to AC</td>
</tr>
<tr>
<td>ADD 1xxx</td>
<td>9xxx</td>
<td>ADD memory word to AC</td>
</tr>
<tr>
<td>LDA 2xxx</td>
<td>Axxx</td>
<td>Load memory word to AC</td>
</tr>
<tr>
<td>STA 3xxx</td>
<td>Bxxx</td>
<td>Store contents of AC in memory</td>
</tr>
<tr>
<td>BUN 4xxx</td>
<td>Cxxx</td>
<td>Branch unconditionally</td>
</tr>
<tr>
<td>BSA 5xxx</td>
<td>Dxxx</td>
<td>Branch and save return address</td>
</tr>
<tr>
<td>ISZ 6xxx</td>
<td>Exxx</td>
<td>Increment and skip if zero</td>
</tr>
<tr>
<td>RRE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLA 7800</td>
<td>Clear AC</td>
<td></td>
</tr>
<tr>
<td>CLE 7400</td>
<td>Clear E</td>
<td></td>
</tr>
<tr>
<td>CMA 7200</td>
<td>Complement AC</td>
<td></td>
</tr>
<tr>
<td>CME 7100</td>
<td>Complement E</td>
<td></td>
</tr>
<tr>
<td>CIR 7080</td>
<td>Circulate right AC and E</td>
<td></td>
</tr>
<tr>
<td>CIL 7040</td>
<td>Circulate left AC and E</td>
<td></td>
</tr>
<tr>
<td>INC 7020</td>
<td>Increment AC</td>
<td></td>
</tr>
<tr>
<td>SPA 7010</td>
<td>Skip next instruction when AC is positive</td>
<td></td>
</tr>
<tr>
<td>SNA 7008</td>
<td>Skip next instruction when AC is negative</td>
<td></td>
</tr>
<tr>
<td>SZA 7004</td>
<td>Skip next instruction when AC is zero</td>
<td></td>
</tr>
<tr>
<td>SZE 7002</td>
<td>Skip next instruction when E is zero</td>
<td></td>
</tr>
<tr>
<td>HLT 7001</td>
<td>Halt computer</td>
<td></td>
</tr>
<tr>
<td>I/O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INP F800</td>
<td>Input character to AC</td>
<td></td>
</tr>
<tr>
<td>OUT F400</td>
<td>Output character from AC</td>
<td></td>
</tr>
<tr>
<td>SKI F200</td>
<td>Skip on input flag</td>
<td></td>
</tr>
<tr>
<td>SKO F100</td>
<td>Skip on output flag</td>
<td></td>
</tr>
<tr>
<td>ION F800</td>
<td>Interrupt on</td>
<td></td>
</tr>
<tr>
<td>IOF F400</td>
<td>Interrupt off</td>
<td></td>
</tr>
</tbody>
</table>

3 Implementation of Simulator

Integrated Development Environment (IDE) is equipped with a code-editor and a built-in assembler. It provides primary user interface. The code-editor provides many commands like disk I/O, cut, copy, paste, find and replace etc. This is a Single Document Interface (SDI) implemented through Windows® standard Graphical User Interface (GUI). The IDE is fully equipped with Windows® standard help system, which provides index based help, keyword search, context sensitive help and “what is this?” help.

The assembler is built into the IDE and can be invoked directly from IDE. Figure-3 represents the assembler. This is a 3-Pass assembler with full error detection capabilities. It uses a message-pan (part of the IDE) to pass messages to the user. Comments and extra white spaces are first removed from the source code. Pass-1 carries out syntax analysis through finite automata. Pass-2 generates the address symbol table. Pass-3 finalizes the assembly process and generates the target machine code. The error handler remains intact with all three passes and address symbol table accompanies the pass-2 and 3. Code is first prepared to be acceptable for assembler. Figure-4 (a) and Figure-4(b) present the flowcharts of this process:

![Figure-3: Block diagram of assembler.](image-url)

![Figure-4(a): Flowchart of code preparation process](image-url)
The next step is syntax analysis. The syntax analysis is carried out through automata. The automata below present how the different types of symbols are defined and validated.

Automaton in Figure-5(a) defines and validates Label Symbol. A Label Symbol consists of 2 to 4 characters. First character must be a letter and last character must be a comma (,). Rest of the characters (if any) are either letters or digits. No other character is allowed in a Label Symbol. The automaton consists of six states ranging from L1 to L6, L1 being initial state and L6 being the final state. An input character changes the state. For example if we are on state L4, only a letter can take us to L5. In this automaton l represents “letter”, d represents “digit” and operator | means exclusive-or. Therefore \((l | d)\) means either letter, or digit, but not both. The only symbol is validated that takes us from initial state to the final state.

Automaton in Figure-5(b) defines and validates the decimal operand symbol, which is a 16-bit signed integer (-32768 to 32767). Obviously the symbol can contain at most 5 decimal digits and an optional (+) or (-) sign. In this automaton, d represents “digit”.

Automata in Figure-5(c) and Figure-5(d) define and validate different instruction syntaxes. In Figure-5(c), automata for every syntax variation are presented. We combined these automata to form a single automaton (Figure-5(d)) capable of defining/validating all variation.

In Figure-5(c) and Figure-5(d) the abbreviations mean as follows:

- L: Label
- MRI: Memory Reference Instruction
- LO: Label Operand
- I: mode symbol ‘I’
- NMRI: Non-MRI
- ORG: ‘ORG’ pseudo instruction
- END: ‘END’ pseudo instruction
- HEXA: Hexadecimal Address
- DEC: ‘DEC’ pseudo instruction
- DECO: Decimal Operand
- HEX: ‘HEX’ pseudo instruction
- HEXO: Hexadecimal Operand

Another important component of the product is virtual machine (VM). The idea of VM in this project disagrees (slightly) the theoretical idea of VM. The reason is that the basic theme of the product is not the VM, rather the product intends to simulate the complete features of a programmable general purpose digital computer. To implement this theme we need something like the VM so that we could be able to demonstrate all features of conceptual Basic Computer using the Intel® architecture. As the software begins to execute on Intel® computer, it begins to behave like Basic Computer. It allows executing a machine code quite different from that is for Intel® architecture. The assembly language of Basic Computer is quite different from the assembly language of Intel® architecture. Since all these characteristics are like a VM, therefore we can rightly call this component a Virtual Machine.

It is worth illustrating here how the virtual machine logically exists within the system. Figure-6 depicts it:

The Program Tracer demonstrates statement-by-statement execution of program. Since this module is aimed at visualizing the system state on every statement execution, the association of visual controls with the virtual components (Registers, Flags and Memory Unit) is a compulsion. Methods have been written for every virtual component, which provide a dual interface: the user interface which is concerned with updating visual controls on the screen, and the system interface which is aimed at updating the virtual components. Figure-7 shows how Program Tracer works:

The functioning of Program Tracer is as follows:

- **Startup**: A procedure Startup executes first, which loads the program from CodeListing into virtual memory unit, initializes program counter PC to program entry point and sets register set and flag set to initial values. It updates every visual control associated with some hardware component accordingly and hands over the charge to Trace Controller.

- **Trace Controller Triggers**: The trace controller can be triggered manually or automatically at constant intervals. When trigger takes place, the complete execution cycle is initiated and completes its steps: Fetch, Decode and Execute. It first Fetches the instruction code at memory location pointed to by the PC and places it in instruction register IR. The instruction is then checked. If it is HLT program terminating instruction, the tracing process stops.

- **Execute Instruction**: If the instruction is other than HLT, the operation specified by the instruction is performed, the virtual hardware components and associated screen controls are updated, and execution stops to wait next trigger. This way the step by step execution of program is performed.
Figure-5(a): Automaton to define and validate label symbol.

Figure-5(b): Automaton to define and validate decimal operand symbol.

Figure-5(c): Automata for different types of instruction

A. Memory Reference Instructions

B. Non Memory Reference Instructions

C. ORG and END Pseudo Instructions

D. DEC Pseudo Instruction

E. HEX Pseudo Instruction

Figure-5(d): Compound automaton of Figure-5(c)
The Program Tracer fully visualizes the impact of every instruction on the system components, like registers, flag etc. Registers and flag contents are visible on the screen. The memory unit is also on the screen. The contents in registers and memory are displayed in binary, hexadecimal and decimal number format to facilitate user understanding. A built-in help system remains vigilantly intact with the tracer to guide user instantly.

The CPU Tracer is one of the most complex modules of the project. The CPU Tracer offers clock based simulation. That is, if an instruction requires 6 clock pulses to complete execution, the CPU tracer will execute it in 6 steps. This simulation mode exhibits the lowest level of operation of the Basic Computer. The execution of a machine instruction requires many micro operations. Design of certain computer allows some of the micro operations to take place simultaneously and other micro operations require exclusive execution. This means one or more micro operations can be carried out on a single clock transition. The CPU Tracer implements each and every feature of the hardware components used in Basic Computer. The CPU tracer works as given below:

A procedure Startup carries out all initializations required to run CPU tracer. It does following tasks:

- Loads machine instructions code from CodeListing to virtual memory unit
- Populates visual controls: memory grid with memory contents and listing grid with code listing.
- Sets PC to program entry point.
- Initializes virtual hardware components to initial states.
- Calls procedure ActivateCU that performs CU’s functionalities and issues control signals as required by the microoperation to be carried out.
- Updates visual controls on screen by calling a procedure RefreshScreen to reflect current system status.

The dotted line in Figure-8 below shows the scope of procedure responsible for performing all action associated with a clock pulse:

Following happens on each clock pulse:

- Check start-stop flag \( f_S \); if it is equal to 0, the program (under simulation) should terminate, else there’s an indication that the program has to go on with some executable instructions.
- If \( f_S \) is not equal to 0, then call a procedure Process that implements the ALU and carries out the process required by the microoperation.
- Then a procedure Execute is called which updates all system components by moving data along them, as specified by the microoperation under execution.
- Again the ActivateCU is called which prepares system for next microoperation by updating the control signals according to microoperation next to execute.
- The procedure RefreshScreen is again called to reflect recent system status on the screen.
- Loop back for next micro-operation.

To demonstrate I/O operation, the product is equipped with one input device: the keyboard and one output device: the printer. These devices have been virtually implemented. The virtual keyboard provides basic character set. The virtual printer has many simulation controls. It can be configured to read and print a character automatically at specified interval, otherwise as per user intention. These simulation controls provide user the facility of concentrated observation.
4 Testing

4.1 Modular Testing
The product is composed of many modules. Each module, in turn, consists of many functions and module-related data. There are also functions, objects and data items in the program with global scope. The overall functionality of the product is a result of mutual interface of modules. The modules interact with each other either through module-functions or through the global data elements.

The entire program consists of many modules and a global area. The global area contains the global data, global functions and global objects. Everything in global area can be accessed from everywhere in the program. These global components provide mutual interface among the program components.

The modules consist of module-data and module-functions. The data in the modules are private. The functions in the modules fall into two categories. The module-specific functions are concerned with the module which these belong to, and are private to their modules. The interfacing-functions are aimed at providing mutual interface among the modules, and are public functions. The functions may have their local data, which they use for their own purpose.

After this brief description, now we are able to talk about how we tested the product at module level. The testing at module level was a simultaneous process, that is, every function in module was tests soon after it was completed. On completion of the entire module, the testing of module was conducted as a whole. The entire product consists of four logical modules: The Editor, the Assembler, the Program Tracer and the CPU Tracer. These components are further composed of sub-component. We adopted the above mentioned paradigm to test these modules.

4.2 Integrating Testing
After modular testing, we integrated the modules to form a single product. This process introduced many errors that came into focus as a result of integration. Therefore we conducted comprehensive integrated testing.

5 Conclusion And Future Work
There is still a little possibility that the product is left with some bugs. We are of the opinion that further confirmation requires Beta Testing. This is still pending and is left as future plan. Since the product is “Volunteer” in nature, we also suggest and schedule to present the software to potential users to seek their opinion.

As far as the utility of the product in classroom is concerned, we have tested it on a limited scale. We handed over the product to selected students and teachers.

They appreciated it and remarked that it would serve as the innovative and modern teaching approach, which would enhance the productivity of class work many times.

This software product gained considerable appreciation from the teachers and students of computer science, which encouraged us to keep on working with the idea and develop products of the same scope for other computer science courses like Data Communication, Automata Theory, Logic Design, Computer Graphics etc.

We intend to introduce further versions of Computer Simulator, which will simulate more than one computer models. Our recent intention is to include in Basic Computer Simulator a Micro-programmed Control Unit model. Our next attempts will concentrate on real models like Intel® and Motorola® microprocessors.

Figure 8: Flow Chart For CPU Tracer

```
START

STARTUP

• Load machine code from CodeListing to virtual memory unit
• Populated memory grid and listing grid
• Set program entry point to first instruction of program
• Initialize registers, sequence counter and flags
• Activate CU to prepare system for execution of required microoperation.
• Update interactive screen elements with system state

CLOCK TRIGGERS

Is=0

Yes

No

Move result of processing and other data to relevant virtual components as specified by the microoperation

Activate CU to prepare system for the execution on next microoperation on next clock pulse

Update visual components on the screen with current system state

Activate ALU to perform processing specified by required microoperation

STOP
```

Figure 8: Flow Chart For CPU Tracer
References