

# Project Plan

Final Year Project 2016: Simulation of Simple Computer System for Teaching

Wong Yin Lok (Braden)

Supervisor: Dr. K. P. Chan

## Table of Content

Project Background .....	1
Objectives .....	3
Scope and Deliverables .....	4
Methodology .....	6
Challenges .....	7
Schedule .....	8

## Abstract

This Final Year Project – Simulation of Simple Computer System for Teaching is about the development of a simulator of the CPU and memory system, for the purpose of teaching and learning in the Computer Science course COMP2120 Computer Organization. The project aims at delivering two simulation components in one simulator program, one for CPU simulation and the other for cache memory simulation. The CPU simulation will display low level details of instruction set operations via animated graphics. The graphics will essentially include a dynamic CPU architecture representation, in which data flow and register updates will be indicated. The development of the memory simulation will start after the completion of the CPU simulation. It will simulate a list of cache memory slots, which will be updated as the input programme proceeds. The simulator will be developed in Java which can then be distributed to teaching staff and students as a standalone Java program to facilitate teaching and learning.

## Project Background

Computer organization and architecture is the very basic knowledge to be learnt, if not mastered, in Computer Science and related studies. The study of computer organization explores the fundamental functions performed by any computer system – to move data around and perform simple arithmetic operations. These two simple functions, however, are not very intuitive to be programmed. As the operations are carried out by the CPU, which identifies only bits of 1 and 0, all the instructions to be programmed have to be translated to segments of numbers to indicate specific operations and data location according to a scheme referred to as the instruction set. Figure 1 shows an example program of CPU execution.

LD	P0,R4	0000:	06001104	0000003c
LD	P1,R1	0008:	0600ff01	00000040
MOV	R1,R2	0010:	05010002	
LD	P2,R3	0014:	0600ff03	00000044
L: ADD	R4,R1,R4	001C:	00040104	
ADD	R1,R2,R1	0020:	00010201	
SUB	R3,R1,R5	0024:	01030105	
BNZ	L	0028:	0802ff00	0000001c
ST	R4,P	0030:	0704ff00	00000048
HLT		0038:	09000000	
P0: .WORD	0	003C:	00000000	
P1: .WORD	1	0040:	00000001	
P2: .WORD	A	0044:	0000000a	
P: .WORD		0048:	00000000	

Figure 1. simulation of CPU execution

In the course COMP2120, students learn and programme with the instruction set, as well as learn about the basic of memory cache. This means they will have to write code in series of digits and try to keep track of the actual data movements and operations based on these abstract lines of code. Not only is the code written in hexadecimal numbers or symbolic operation codes difficult to follow, the thorough understanding of actual CPU operations is not reflected in the code.

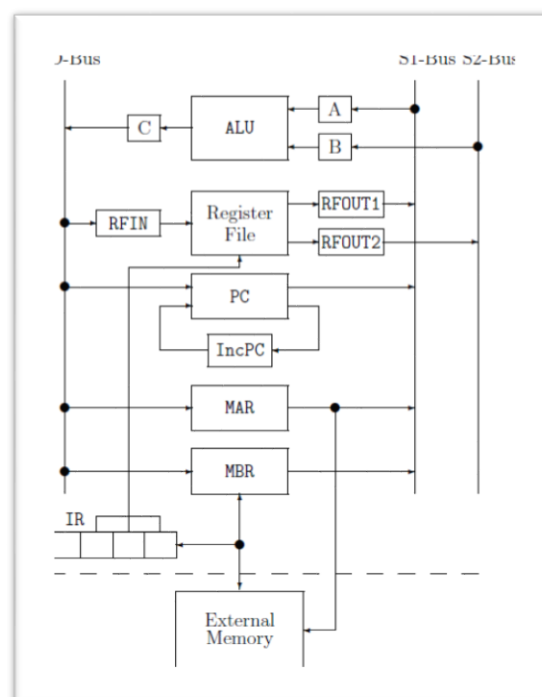


Figure 2. CPU architecture used in COMP2120

Figure 2 shows an example CPU architecture used in COMP2120. In actual CPU operations, each operation code in the instruction set, e.g. ADD and SUB, is broken down to a series of interactions between the CPU components. For example, in an ADD operation with external memory, data is first fetched from External Memory to MAR, moved via S1-bus, supplied to ALU and redirected to Register File update. These details are not shown in the programme written but are crucial to the understanding of CPU architecture and functions.

In addition, memory cache replacement can be a confusing process in which different slots of the memory cache is swapped out and replaced by new memory content. While the number of slots and the size of the memory cache are fixed, the replacements come in different orders, targeting different slots every time depending on the replacement scheme. This is, as what we see with CPU operations, difficult to follow without a proper visualization. To help students better picture the above details, the idea of a graphical simulator is born.

This project is actually not new. Last year, another student took up this project as his Final Year Project, a screenshot of it shown in figure 3. However, the simulator from the project last year serves mainly as a step by step graphical display of CPU operation in a “programme-wise” manner, i.e. showing the steps taken in the programme. CPU components that are involved in an operations are high-lighted sequentially in the CPU display on the left, while contents in the register, cache and main memory are updated accordingly on the right. The simulator, however, fails to demonstrate explicitly the interactions between the CPU components in an “operation-wise” manner. The goal of the simulator, regarding CPU simulator, is to demonstrate the sequence and details of operations taken by different components of the CPU and the interaction of them per operation. While the memory cache simulation can present the update quite clearly, the CPU simulation is subject to improvement.

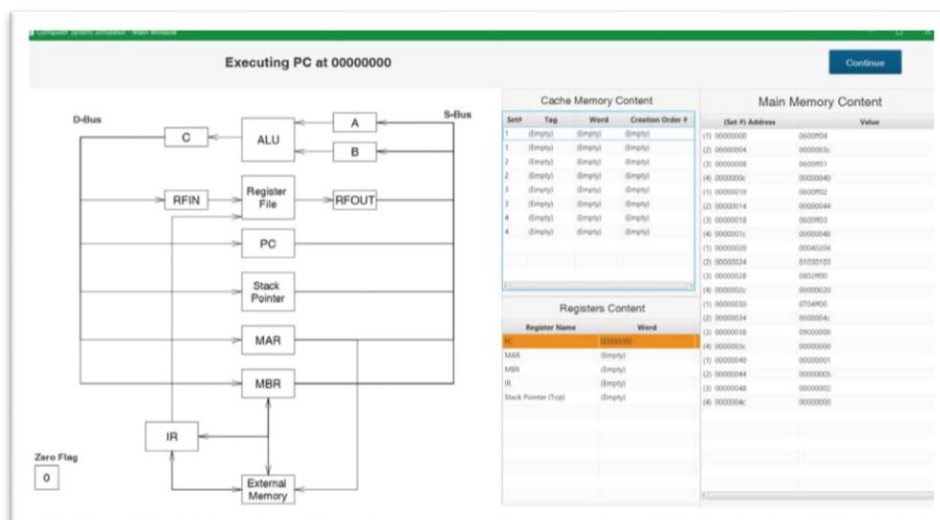


Figure 3. screenshot of the simulator from previous project

## Objectives

The goal of the project, and the purpose of developing such a simulator, is to help students visualize and thus more easily understand the actual operations happening inside the CPU when programming with instruction sets. On the other hand, teaching staff could utilize the tool to build representations of examples or questions and convey ideas about abstract concepts and tedious executions to students more concretely.

The simulator will serve as a learning and experiment tool besides textbook and lecture slides that students could have hands-on experience with. Students will be able to use the

simulator to replicate the CPU architecture and logic taught by the teacher, and to verify principles and theories by trying different configurations on the simulator. In correspondence to this, teaching staff could simulate CPU architecture with this simulator as a supplement to content in lecture slides. They could demonstrate the principles or operations taught with the simulator.

The simulator will also act as a point of reference and possibly a tool for course assignments. After writing lines of cumbersome code on instruction set execution, students could validate the results of their theoretical program by replicating the same configurations and operations on the simulator to see if the expected result is successfully simulated.

## Scope and Deliverables

The simulation of computer system can cover a broad range of components and be complex to be developed. However, given the time constraint, the scope of this project is carefully controlled so that essential and prioritized functions can be delivered.

This project follows tight the three-phase development paradigm proposed by the department. Phase 1 is the Inception phase. This project plan and a project web page loaded with details about the project shall be delivered. The project website will contain information about the project in general, including title, responsible student, supervisor and project description. It will also be updated with project progress, upon achieving milestones or significant activities. The website will also be loaded with related documentation, including this project plan initially, and the future interim and final report. Lastly, the website will contain a section detailing the results of the project. In short, the website will be self-complete and provide all the information one should know about this project.

Phase 2 is the Elaboration phase. The completed CPU simulation will be delivered in this phase. While the CPU simulation will be delivered first in Phase 2, the final CPU and memory cache simulator will be a single Java program serving both simulations. There will be added features to cater the missed objectives, and removed components to make the simulator more focused.

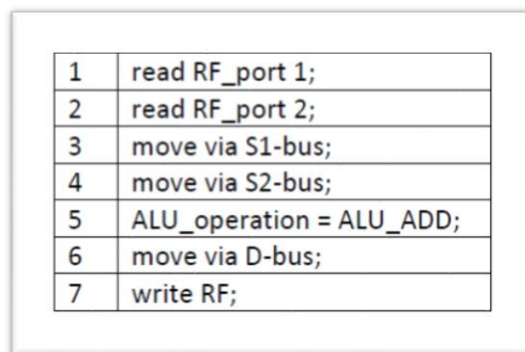
To understand the details of the delivered implementation, one should have understanding on the concepts of instruction sets and memory caching. An example of programming with instruction set is shown in figure 1. Instruction set is a set of operations, including common ones like ADD, SUB and MOV that is performed by the CPU. The pseudo code is written in names of operations and data registers like “ADD R4,R1,R4” while the actual code, in assembly language, is like series of hexadecimal numbers corresponding to different operations and registers or memory address.

However, simply understanding the assembly language and pseudo code to hexadecimal numbers translation does not guarantee understanding of how CPU functions, as a single operation is implemented by a series of data movement and operations among the different modules in the CPU. To facilitate this understanding, the CPU simulation will provide the features as follows.

To begin with, the CPU simulator will contain 3 major parts. The first part will be animated graphical display. This graphical display is essentially a video player that shows data movements and operations in the CPU. The CPU architecture used throughout COMP2120, i.e. figure 2, will be the background of the display. When the play button is hit, there will be colored points, representing data signals, moving on the CPU image to indicate data movement. Relevant CPU modules on the path, e.g. MAR and PC, will have content update which will be shown in other description fields on the right beside the display.

The second part of the CPU simulation will be programme input and configuration. The simulator will provide a large text field for users to type in their programme in hexadecimal numbers. Since the reference from numbers to operations is needed for the simulator to interpret the programme, e.g. "00000001" refers to "ADD", there will be a list of reference matching for users to configure after entering the programme code. This programme input will serve as the basis for the graphical display to run.

The third part will be operation code definition. As the objective of the simulator is to allow users to learn and better understand the actual operations inside the CPU, instead of merely entering the programme and watching the graphical display run, users will be able to define the implementation of any single operations. For example, the user will be able to define the operation "ADD" by defining something similar to figure 4.



1	read RF_port 1;
2	read RF_port 2;
3	move via S1-bus;
4	move via S2-bus;
5	ALU_operation = ALU_ADD;
6	move via D-bus;
7	write RF;

Figure 4. sample "ADD" definition

As students are required to programme with custom instruction sets in the course, this part of the simulator allows them to test their very own implementations and enhance understanding of the principle. However, it is not possible for the simulator to support definition of any unseen operations, because for operations that require the use of ALU – Arithmetic Logic Unit – to perform simple arithmetic like addition and subtraction, the ALU operations will have to be pre-defined by the simulator. Therefore, this simulator will only guarantee the definitions of the basic operations listed in table 1, and the corresponding sample definitions will be delivered along with the final simulator.

Phase 3 is the Construction phase. The final tested implementation will be delivered. With CPU simulation completed in Phase 2, the focus of Phase 3 will be memory cache simulation.

Supported Operations
ADD
SUB
AND
OR
NOT
COPY
MOV
LD
ST
BR
BZ
BNZ
HLT

Table 1. Operations supported by the simulator

Memory cache replacement is the process of reading comparably slow memory into fast-access CPU cache. With limited size in the cache, there are a variety of access method and replacement schemes to read and write data to and from the cache. It is not possible to cover all implementations in this simulator, the access method will be limited to 2-way associative mapping and the replacement scheme will be optional in FIFO(First-In-First-Out) or LRU(Least-Recently-Used).

The memory cache will be displayed in an indented list numbered by set number and cache line number. Relevant content will be updated as the input programme proceeds. There will be 2 cache lines in every set and the number of cache sets will be limited to 8. There will be the option for users to select whether the replacement scheme will be FIFO or LRU.

## Methodology

In the first part of CPU simulation, an animated display will be built. While the display background will be loaded with the CPU image, the entire frame will be updated millisecond by millisecond to create animation effect. The animation update will be programmed with the Java Graphics package, which provides a lot of functions on drawing basic shapes and formulating graphic movements.

Operations defined by users in the simulator will be formulated as function objects. These function objects will then be referenced in configuration with the input programme and to load the graphical display.

Memory cache replacement in this simulator will be functionally the same as that in the previous simulator. A portion of code on memory cache simulation may be able to be referenced or even recycled. Based on this previously developed code, the plan is to build wrapper functions that encapsulate the usable codes as a black box, and feed the black box with inputs or commands to get desired outputs that can fit to this simulator.

Development of memory cache simulation will start after the completion of CPU simulation. As the simulator will be programmed in Java, most code will be encapsulated into objects that are easily expandable by building auxiliary functions to interact with the objects or modifying the attributes in the object definition. Memory cache simulation can thus be developed after CPU simulation without the need of large-scale amendment to the existing code.

The final version delivered will be tested. Testing of the simulator will include sample definitions of the instruction set in table 1. These operations will be defined and referenced. A sample programme containing all of these operations will be loaded and the graphical display will be run to validate results.

## Challenges

The major challenge of the project lies with the development of dynamic, animated graphics. As development of graphic requires special attention to the positions of different graphical objects. And it can be an exhausting process to figure out and programme the correct coordinates of the boundaries and the vertices of the objects in every time frame, let alone the difficulty of drawing attractive graphics by setting the parameters in code.

To address the problem, the use of Java as the main programming language in this project is determined. As Java includes a rich library of GUI functions, the difficulty in programming the animation will be alleviated with the use of this language. Moreover, some graphics will be designed with software such as Photoshop and be imported for the use in Java so that the complicated, attractive graphics will not need to be composed solely in code.

The other challenge is the complexity of the project. As simulation of a computer system, no matter how simple, is not easy. It involves thorough understanding of the lower level details of computer architecture and implementation of such. Additionally, with the desire of implementing animated GUI, the time and effort needed for the project could end up deviating from expectation and pose great threat to meeting submission target.

The project plan is made flexible to a certain extent in order to tackle the challenge. The development of the CPU simulation is expected to be the most complex part in this project and will be delivered in early stage. There will be room for adjustment in schedule if the CPU simulation does not finish on time. The development could go on while the time allocated for the less demanding cache simulation will shrink.

## Schedule

The following is the timetable of the project proposed by the department.

Time	Event and Deliverables
2 October 2016	Completion of Phase 1 Inception <ul style="list-style-type: none"><li>- Project plan</li><li>- Project web page</li></ul>
Early-January 2017	First presentation
22 January 2017	Completion of Phase 2 Elaboration <ul style="list-style-type: none"><li>- Preliminary implementation: CPU simulation</li><li>- Interim report</li></ul>
16 April 2017	Completion of Phase 3 Construction <ul style="list-style-type: none"><li>- Finalized tested implementation: Memory simulation</li><li>- Final report</li></ul>
Mid-April 2017	Final presentation
2 May 2017	Project exhibition