CM1205: Architecture and Operating Systems

STRUCTURED PORTFOLIO **(100%)**

Theo Baur | C21050251| 10/05/2022

WEEK 1: INTRODUCTION

Reflection on Learning Materials

This was my introductory week with CM1205 and because of this I am not able to have a seminar discussion. I found this week interesting as it gave me insight into some of the major motivators behind all computer development as well teaching me some of the earliest forms of computing, which I did not consider as computers before

I learned that one of the major motivators for the development of any computer system is the inherent laziness of people, this has been consistent for all computing systems that I have learnt about the week. A specific example of this could be the invention of Napier's Bones by John Napier in 1617, this enabled the multiplication and division of numbers to be expressed in terms of addition and subtractions in a much simpler way. This meant that complex multiplication with large numbers of digits could be performed very quickly. John Napier was a mathematician and a physicist therefore it seems as if Napier was too lazy to perform complex multiplication repeatedly if it took too long, it follows that due to his inherent laziness he developed a computer to help solve maths problems with less effort. This idea is important to me as it should influence my programming so that I design systems that ultimately make life easier.

I also learned about the idea that practice often comes before theory and understanding. In terms of computing this means that we often create computing systems to use for a certain purpose before understanding how the system itself fully works. Some examples of this are early humans using a counting systems with their fingers before understanding number systems. The first Turing-complete computer, the Analytical Engine, was created by Charles Babbage in 1833 for arithmetic, whereas the idea of Turing-completeness *(Turing, 1936)* was only created in 1936 along with Alan Turing's seminal paper. This should help serve as a reminder that you can create useful programs whilst not having a complete knowledge of the field. For example, whilst I may not understand programming at the lowest level, I am still able to create programs with higher level languages such as Python

Whilst reading the section about Alan Turing, I thought it would be useful to refresh my memory on how the most fundamental Turing Machine works, as this is the basis for modern computers with memory. I chose to watch "Turing Machines" *(EngMicroLecutures, 2013)* which helped to update me on how exactly machine states worked.

WEEK 2: STRUCTURE OF COMPUTERS

Reflection on Learning Materials

This week the content was focused on the lowest levels of computer architecture, including basic electronic logic gates and how these can translate into arithmetic functions which in turn can explain how the computers work at the most basic level. I found this week's material fascinating as I have spent a lot of time focused on higher level computing such as coding using python and web development, but I did not have a good understanding of how higher-level computing translated down into the lowest levels.

My favorite part of this week's slide was the expansion on logic gates, whilst I did previously understand how to use logic gates, I was not aware of the important role they play in the CPU. For example, the Arithmetic Logic Unit, which is used in the CPU to perform arithmetic is divided into arithmetic circuits such as the *decoder* and *full adder* which can be further simplified into simple logic gates such as NOT, AND, OR and XOR. I learned these are used as logic gates take two inputs and return an output, these two inputs can be represented by two different voltages on a circuit, which can represent binary. This helped to link together knowledge of specific topics for me.

Most computers follow a Von Neumann model as the model supports sequential processing, where tasks are performed in the order they are received, however struggles in performance as there is a bottleneck inherent to the model. There have been some Non-Von Neumann models such as quantum computers which abandon sequential processing and use parallel processing however all of these are still in development stages and are not used in the mainstream.

Reflection on Seminar Discussions (remove this if not applicable)

In this week's seminar we needed to explain the history of computers to an elderly woman in the form of a one-to-one meeting. The woman only has secondary school education from around 50-55 years ago and enjoys Reading (crime, fantasy fictions), travelling, gardening, knitting and history and lives in a relatively rural area.

I discussed this with my group, and we immediately agreed that we would need to discuss any historical figures and their discoveries in a very simple matter, and they should be listed in chronological order as the woman is used to learning about history this way. Someone in my group also decided to leave technical explanations such as how Napier's Bones *(Napier, 1617)* as this would not be an efficient use of time and is unlikely to be understood. Another idea that was discussed was to explain the history like a story as she enjoyed reading books. We took extra care to try and keep the woman engaged by telling some trivia about Alan Turing such as his contribution to the *Enigma* as the woman enjoys fiction, which contains engaging stories and the movie *The Imitation Game* was based on this event.

WEEK 3: ASSEMBLY LANGUAGE (OVERVIEW)

Reflection on Learning Materials

This week we learned about the basic's aspects of Assembly Language, as well as performing an exercise. I already had some elementary knowledge from my GCSE course about the justification for using Assembly Language however a lot of it had been forgotten, so it was useful to refresh my memory on what exactly it was.

One of the most useful aspects of this week's slideshow was that it displayed the advantages and disadvantages of Assembly Language. Most of the negative points revolved around the impracticality of coding with this low level of a language. It has much less readability than higher level counterparts making it harder to debug, it is also harder to understand and learn due to its low level of abstraction from the computer hardware. However, these are to be expected from a lower-level language. Some more interesting disadvantages that I had not previously known was that it is not portable, meaning that a program written in Assembly Language for one machine is unlikely to be able to run on any other machine and OS. Also, to get a more efficient program it is often more advantageous to focus on improving the logic of an algorithm (which can be done in a higher-level language) than improving the assembler.

I also learned that one major use of Assembly Language/machine code was reverse engineering, especially in the field of anti-malware and I would also assume game emulation. This is because in situations where you do not have access to source code you will often have to disassemble a program which is only given as a .exe in some sort of machine code. This needs to be translated into a higher level to fully understand what the program does, and in the case of malware prevent it from spreading *(Ortolani, 2018)*.

Reflection on Seminar Discussions (remove this if not applicable)

This week's seminar involved trying to inform a frustrated customer that a Core i5-8265U machine will perform better than an i7-3517U machine when the customer doesn't think so. It seems as if the customer is under the impression that the i5 and i7 do not make a significant difference in performance.

I had discussed this problem with a partner, and we agreed that it is important to discuss a few major points: generation, number of cores, clock speed and misconceptions. We both agreed to explain 2 of these points. I explained that the digit after the prefix i5 or i7 is the generation of the CPU (when it was built). Generally, the higher the generation, the smaller the transistor size and power usage, therefore more transistors can be put into a CPU. This means that the performance of a CPU model (e.g. i3, i5, i7) will normally improve over time. In this case we are comparing an 8th generation i5 to a 3rd generation i7. My partner then explained that in general the number of cores in a CPU will lead to a performance increase as a multiple step task can be split among cores simultaneously. In this case the i5 has 4 cores and the i7 only has 2 cores. He also added that a higher clock speed can sometimes lead to a higher performance as a CPU has a certain number of actions that it can complete per cycle, and if it has a higher clock speed it can perform more of these cycles per second. However, I reminded him that the number of actions

that a CPU can complete per cycle is different for each CPU, therefore clock speed is not accurate. Finally, I went on to explain some misconceptions as if the customer hears certain claims, she might get the wrong impression again. For example, in each generation of CPUs the order of performance will go from i3, i5, i7, i9. Therefore, they may think that because the i7 is better than the i5. However, this is only applicable for when comparing a single generation of CPUs. Therefore, we can conclude to the customer that the i5 will outperform the i7.

WEEK 4: ASSEMBLER AND INTEL 8086 ASSEMBLY LANGUAGE

Reflection on Learning Materials

This week we went into detail about Instruction Set Architecture in the context of preparing to learn basic assembly language. Whilst knowing some of the basic terms, such as registers or microarchitecture such as the Arithmetic Logic Unit, it became clear as I advanced through the slides that most of this topic was entirely new content to me. In general, this week's topic was very interesting to me as understanding the fundamentals of a CPU and how registers function in detail was never explained to the lowest level in my previous computer science classes, however in this topic we learn how to use Assembly Language to control registers.

Perhaps my favorite sub-topic was the technique *byte-swapping*. For non-Intel systems, when storing a number that is larger than 8 bits the most significant byte, the largest part of the number, is stored before the less significant bits. This is known as *Big Endian*. However, for intel systems the number is stored in the opposite way, known as *Little Endian*. What was fascinating to me was that this small practice that started early on meant that Intel and non-Intel systems have been separated and we are now unable to change this as the systems have both grown too complicated.

After learning about *byte-swapping* we went into detail about what is stored in a register. There can be specialized registers or general-purpose registers. For the Intel 8086 some general-purpose registers are EAX, used in multiplication and division, or the ECX which can be used as a counter when performing a loop. There are many more examples of general-purpose registers however seeing these examples of them helped me get a better understanding of what different types of registers do.

WEEK 5: ADDRESSING MODES, LOOPS AND JUMPS

Reflection on Learning Materials

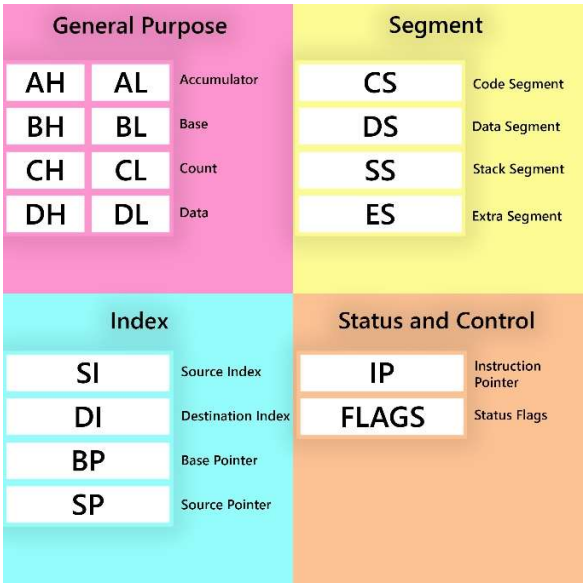
This week's learning materials focused on the methods of addressing as well as the structure of loops and jumps in assembly language. I enjoyed studying how different modes of addressing works as it helps provide context to how data is moved across registers in detail.

There are several different forms of addressing: Immediate, Direct, Register, Register Indirect, Indexed, Based Indexed and Stack. These range in simplicity from the simplest, Immediate, where the instruction is only to load a register to a set value which is useful when using constants, to more complex instructions such as Indexed and Based Index addressing. Indexed Addressing is used to access record structures, an instruction may be to load a register with the data at another register plus x bytes. The number of bytes specified will correspond to a piece of data in this second register, if the data is structured as a record. Based Index Addressing is similar however it requires two registers, the first acting as a base and the second acting as an index to specify a value.

Learning about different types of addressing made it clear to me that all types of addressing are useful but in their own specific contexts. Understanding exactly how each one functions is very helpful to try and identify each context for itself. This should hopefully make any code I choose to write in assembly language more efficient, however this concept of understanding contexts should help me to write better programs in general.

Reflection on Seminar Discussions (remove this if not applicable)

This week our groups were tasked with creating an infographic for an online course that shows the registers in an 8086 microprocessor. The reason an infographic is useful in this case is because there is a lot of content that may be understood or remembered better if represented visually. For this I used Photoshop and the rest of my group used Microsoft Publisher and Word.



After discussing this with my group all of us collectively decided that we should represent the different categories of registers separately. This included the general purpose, segment, index and status and control registers. To better visually separate the sections one member of my group decided to color-code each section, I decided to apply this to my work. My other group member gave me some constructive criticism on my infographic and said that there could be more text explaining the function of each register in the future.

WEEK 6: PROCEDURES AND SUBROUTINES, STRING INSTRUCTIONS

Reflection on Learning Materials

This week the content was centered around procedures and functions, and strings. The slides covering procedures and functions were engaging to me as I already have some familiarity with how functions work as they are part of most programs that I have written, this means that the ideas were less difficult to follow than the strings section.

As mentioned, the bulk of how functions work was knowledge I already had however it was useful to have functions explained using the stack. When a function is called the address of the instruction in the function is put onto the top of the stack, executing the function, and when the return is reached, the address from the next instruction is removed from the stack meaning it can branch back into the main program. We have not covered the concept of the stack as much however I found it interesting as I had used functions so much in most of my programs, yet I did not know how exactly they interacted with the CPU.

After learning about procedures, I focused on the slide teaching about strings. Here I learnt about the special types of operations that can be used on strings, ranging from REP which repeats a string until the ECX (loop counter) becomes 0 to CMPS which can compare two strings are equal.

Overall, these two topics helped provide a deeper understanding of two very familiar concepts: functions and string, by explaining how they work in terms of registers.

Reflection on Seminar Discussions (remove this if not applicable)

Last week we wrote some code in assembly language and compared it with our partners and gave some feedback on the code.

For the first exercise our code was identical as the structure document we had been given was the same and we had been given instructions on what to do in the first exercise. Both of our programs worked for this section, so no real feedback was needed. This was also the case for the second question where both of our loops worked correctly and were identical.

It was worth noting that whilst I was writing the code, I was shown a syntax error several times as I was compiling the template code in 64 bit rather than 32 bit.

The input and output merged program also functioned as it was meant to so there was not much individual feedback to be given. However, some things that I picked up on were that my folders weren't structured very well as I was keeping my .asm files in the downloads folder instead of keeping them in a more secure area. Whilst I was working, I also was not able to view readConsole.asm and writeToConsole.asm at the same time as debugging the exercise 3 .asm file as the main methods were interfering with each other.

WEEK 8: MEMORY, SECONDARY MEMORY, I/O, BUS

Reflection on Learning Materials

This week I learned about memory and secondary memory. I already had a good understanding of different types of memory such as caches, RAM and hard drives and their respective functions however I ended up learning a lot about how a hard disc functions in terms of hardware. I also gained some interesting insight into why caches were created.

The most interesting area to me was the inherent problem with how CPUs interacted with memory and how this led to the creation of caches. A CPU will not get a word for several cycles as the memory is not located physically close enough as it is not inside the CPU, this could be due to cost reasons and physical sizing issues. This means that if a word is referenced in an instruction before it has arrived the CPU will need to stall. Caches can fix this problem as they are a small amount of fast memory inside the CPU. The words that are likely to be referenced can be preloaded into the cache, therefore the CPU does not always need to stall a process. I had learned about the Von Neumann Bottleneck concept in earlier modules, and this has added to my previous knowledge.

Another interesting topic this week was how hard disks store data. I had no idea how this physically worked before. I learned how the pointer can magnetize certain areas of the disk to represent binary which can be read by using a coil which will generate current when a magnetic field changes. Overall, this was interesting to me as I had learned about the concepts of magnetic coils in my physics A-Level class and seeing how that linked into computers was very satisfying.

Reflection on Seminar Discussions (remove this if not applicable)

This week our seminar task was to provide and receive feedback for our assembly language programs. One program was simply adding some text output to last weeks input and output program that asked the user for their name and outputted their name along with a poem. The second exercise was to identify which areas of code were responsible for reading a number as a string and converting into a numeric value and which area converted the numeric value back into a string to output to the console and making these into subroutines. Then we needed to add two numbers together and output them using another subroutine.

The feedback I received was that I performed the task well as my program successfully printed out text asking for the user's name and outputted it to the console along with a poem as per the specification. As the specification was very specific in how they wanted the code to be written there was not much creative freedom to write different code so there were no real improvements that were suggested. For the second area of the code, I also solved it correctly and the structure was virtually identical to my partners, but in conversation I brought up that it took me a long time to solve this method, to which he gave the advice to draw inspiration from other subroutines and exercises to help understand which registers to use.

WEEK 9: SECONDARY MEMORY, I/O, BUS (CONT')

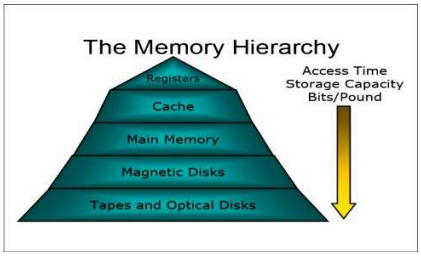
Reflection on Learning Materials

This week was the second half of the secondary memory topic. This was a very content heavy topic and some of the new things that I learned were about Solid-State Drives and CD Drives. This week was mainly technical knowledge about how these devices work which can be interesting in its own respect however it took me longer to grasp this week's content as there was a large amount of content.

I learned that SSDs function in completely different ways to conventional hard disk drives. Electrons are stored in floating gates as either charged or not charged, representing 1 or 0 respectively. These gates are arranged in grids which are called "blocks". A problem with an SSD is that it traps electrons inside cells, whenever the cells need to get rid of electrons (when a cell is completely erased) they go through a process called "tunneling" which can erode the cell and cause corruption. This was something I was not previously aware of.

The other section that interested me was the CD section. I had previously learnt a small amount about CDs in my Physics A-Level which made the subject engaging to me initially. I learnt about how pits and lands can represent binary information and be read off the CD using a laser beam. The basic idea is that when a laser hits a pit, light is scattered due to its curved shape, causing a weak signal representing a 0. When a laser hits a land, light is directly reflected due to its flat shape representing a 1. This added some much-needed depth to my physics knowledge about CD drives.

Reflection on Seminar Discussions (remove this if not applicable)



This week we needed to make some changes to the organizational structure of a company drawing inspiration from data organization.

When joining the seminar this week I pitched the idea that we could structure the company in the form of the memory hierarchy. The memory hierarchy place the fastest and most important storage (to access the CPU) on top and the slowest and least essential to have contact with the CPU on the bottom. An idea of how this can be played out in company structure is that we could say that the CPU represents a CEO or a board of directors that every business decision must go through. The registers can represent the project leaders for each company, the cache can represent the high-level managers for each project leader, the main memory can represent managers of marketing, R&D or sales and the levels below them can be a hierarchy of workers. The idea is that each level is more and more disconnected from the CEO or CPU as it must prioritize tasks to do each day. As I was the first to bring up this idea my peers all agreed that this could be the way to structure the company. Someone in my group brought up the idea of peer-to-peer style structuring meaning that each employee can contribute a small, equally important amount, however we agreed that it would be impractical for any company large enough to require a HR manager.

WEEK 10: INTRODUCTION TO OPERATING SYSTEMS, CPU SCHEDULING

Reflection on Learning Materials

This week we learned about operating systems in depth and different types of structures for computer systems. This included batch, multiprogrammed, timesharing, embedded and distributed systems.

These systems all try to make optimal use of the CPU by not letting it become idle for long. A CPU is idle when it is waiting for an IO device. The batch system was the first variation of computer systems; however, it was inefficient as there were long periods of time where the CPU would idle due to waiting for IO. Another form was the multiprogrammed systems where several jobs were kept in the memory simultaneously. This means that whilst waiting for IO in a process, it could switch to another process causing less idling. Timesharing an extension of multiprogramming, which allows multiple users to use a computer system at the same time.

Embedded systems are computers that operate to perform a small task withing a larger, more complex system. Because they operate in a larger system, an embedded system needs to be reliable and efficient. This means that it needs to consume little electrical power and have a minimum response time to not burden the larger system. Distributed systems are when different systems work together to perform a task. This can involve client server systems, where clients send requests to a central server for a task, or a peer-to-peer system, where peers can act as a server and perform small amounts of a task. This was an interesting topic to me as I was familiar with most of this information already, so the slides acted as a recap.

Reflection on Seminar Discussions (remove this if not applicable)

This week our groups needed to plan to manufacture NAND Flash Memory under the role of a government worker. We needed to briefly say how the chips will be manufactured, what resources are needed, what upstream and downstream industries will need to be approached, what the market is like and how big the technical barrier will be.

We divided each section amongst our group, I was tasked with finding out how the chips would be manufactured, after that we would consult and create the plan together. To manufacture the chip must be designed using CAD tools to lay out the components on each chip, then individual chips are built on silicon wafers in rooms fabrication rooms that are 100 times cleaner than a hospital operating room (*MicronTechnology, 2017*), the wafers are coated in photo resistant material where circuit patterns are exposed on. This stage can take up to 30 days and requires facilities that have Automatic Material Handling to work 24/7(*MicronTechnology, 2017*). They are then assembled by splitting each sheet of silicone into separate chips and placing each chip onto a circuit board and wired using solid gold. This section would be outlined in the plan along with work from the other members of our group.

WEEK 11: PROCESS AND THREADS, PROCESS SYNCHRONIZATION, MEMORY MANAGEMENT

Reflection on Learning Materials

This week we learnt about processes and threads. This topic was completely new to me however after some time trying to conceptualize what I had learned it became interesting to me as it links to other topics, I had learnt such as functions.

I learned that a process has different set states called the running, ready and blocked state. As the CPU can only run one process in one instance there is a queue of ready processes. A process that is being executed is called a running process and a blocked state is a process that requires an event to happen to start running. I then learnt how these processes were represented in an operating system using a Process Control Block. I also now understand how operating systems can swap out a currently executing process with a ready process using context switching

Another area I learnt about was threads, these are a unit of measuring the amount of utilization a CPU has while executing a process. For example, a single threaded process will utilize less of a CPU than a multithreaded process. This helped to explain why multithreaded programming can result in more interactive and economical programs.

Reflection on Seminar Discussions (remove this if not applicable)

This week we had to find out why the "Tragedy of the Commons" (*Hardin, 1968*) problem does not exist in CPU scheduling. The problem outlines that rational actors in a market will ultimately degrade any sort of unhampered resource, contrary to the common good. Using Hardin's example of farmers in a shared field, a farmer will want to maximize gain, therefore he will want to increment one more cow for to sell and we call the positive utility +1. The farmer will do so at the cost of degrading an area of the shared land however the negative utility is a fraction of -1 as the effect is shared amongst all farmers in that field. Therefore, it is in any rational actors' interest to degrade the field. The implication is that processes in a CPU will all overconsume and degrade the CPU beyond the natural levels.

We discussed this back and forth in the group and decided to compare the actors in a market vs actors in a CPU. We collectively decided that the people in a market cannot be compared to the processes in a CPU. This is because we assume that rational actors in a market would want to maximize gain whereas we can't make the same assumptions for a process in a CPU. Processes cannot get extra positive utility from overusing the CPU as they do not sell CPU power on a market, and their internal interest is to get the correct amount of CPU power, any more or less does not affect their wellbeing as they don't have wellbeing and it doesn't provide the process any utility to perform faster or more. We also discussed if a process could physically hog more CPU time than it needs as to do so it would need some sort of arbitrary instruction within the process to do so which doesn't benefit the other instructions in the process as they would all be allocated the exact same CPU time and power.

APPENDIX

EngMicroLectures. 2013. *Turing Machines,* Available at: https://www.youtube.com/watch?v=gJQTFhkhwPA [Accessed 03 February 2022].

Hardin, G. 1968. *The Tragedy of the Commons,* Available at: https://www.hendrix.edu/uploadedFiles/Admission/GarrettHardinArticle.pdf [Accessed 09 May 2022].

MicronTechnology. 2017. *Making Memory Chips – Process Steps,* Available at: https://www.youtube.com/watch?v=M-wNC3Z3ZX4 [Accessed 09 May 2022].

Ortolani, S. 2018. *Reverse Engineering Malware — A Look at How the Process Has Evolved,* Available at: https://www.lastline.com/blog/reverse-engineering-malware/## [Accessed 09 May 2022].

Turing, A. 1936. *On Computable Numbers, with an Application to the Entscheidungsproblem,* Available at: https://www.astro.puc.cl/~rparra/tools/PAPERS/turing_1936.pdf [Accessed 09 May 2022]