Group 27 Final Report - Assembler, GPIO, Extension & Group Reflection

William Springsteen Rory Fayed Stefan Klas Kiran Patel

October 29, 2015

Extension

The extension we chose to do is a game where there are a row of 6 lights, where 5 are red and 1 is green, which will all flash on in order, so only 1 light is on at a time, and when the last one turns off, the first one will turn on again. The aim of the game is to press a button when the green light is lit up. When you do this, the difficulty of the game will increase, which means that each of the lights will be on for a shorter amount of time. The difficulty will reset to the easiest difficulty if you fail on a certain difficulty level. There will also be a score counter, which is simply 3 LEDs that will be arranged in a row and will display your score in binary representation. 3 LEDs for the score counter means that it can display scores from 7 to 0. When you succeed in a level, your score will increase, and when you fail in a level, your score will reset to 0. As part of this extension, we have also extended our assembler to allow the addition of comments to the ARM assembly file.

Design

In implementing our extension, we used 9 LEDs and a button, as well as lots of male-to-female and male-to-male jumper wires, resistors, a breadboard, and a Raspberry Pi. 6 of these LEDs were used for the main game, with each one arranged in a line on the breadboard, and connected to a different GPIO pin on the Raspberry Pi, with each pin set up to be an output pin. To set these GPIO pins up as output pins, we had to write to the control bits of each pin, and make the 3 control bits for each of those pins 001. The pins we used for output were 14, 15, 18, 23, 24 and 25. This meant we had to write a 1 to bits 12, 15 and 24 in 0x20200004 (Control bits for pins 10 - 19, i.e. GPIO Function Select 1 register) and write a 1 to bits 9, 12 and 15 in 0x20200008 (Control bits for pins 20 - 29, i.e. GPIO Function Select 2 register), while all other bits in these memory addresses are set to 0 (Except potentially the reserved bits 30 and 31 in both addresses). Once we did this, we had to clear (Turn off) each output pin before we could use it. To do this, we had to write a 1 to the corresponding bit in 0x20200028 (GPIO Pin Output Clear 0), so to clear pin 18, write a 1 to bit 18 in this register. To then subsequently turn on an output pin, write a 1 to the corresponding bit in 0x2020001C (GPIO Pin Output Set 0). We never actually have to write a 0 to the clear and set registers, and in fact writing a 0 anywhere in these two registers has no effect, as the mere act of writing a 1 will clear/set the pin.

The button we used was connected to an input GPIO pin. We used pin 17 as the input GPIO pin, and we set it up to be pull down, so that the input is high when the button is pressed, and low when the button is not pressed, as we thought it would make more sense for a 1 to mean on, and a 0 to mean off. We can simply check to see if bit 17 in the input detecting register (GPIO Pin Level 0 register) is non-zero to see if there

has been input detected. To set up pin 17 as an input pin, we have to set the last 2 bits of the GPIO Pin Pull-Up/Down Enable (Address 0x20200094) to 01, as 01 is the code for pull down. We also have to preserve the other bits in this register, by using the command `and` on the contents of the register and the negation of 3 (Sets last 2 bits to 0), then adding 1 to the result and storing this back into the register. We then set bit 17 in the GPIO Pin Pull-Up/Down Enable Clock 0 register (Address 0x20200098) to a 1, and then make sure that the three control bits for pin 17 in GPIO Function Select 1 register (Address 0x20200004) are 000, which means pin 17 is an input pin.

The main loop consists of turning on a light, turning off the previous light, waiting for a certain amount of time, and repeating this process with an unconditional branch, until you press the button. The process of waiting involves nested for loops, that both decrement a large number down to zero. The number being decremented can be increased/decreased to make the game easier/harder, as the time that each light will be lit will increase/decrease. During the waiting, we are checking the GPIO Pin Level 0 register, and if bit 17 in this register is non-zero, the button has been pressed. If the button has been pressed, the last light that was lit will stay lit for a few seconds, the difficulty of the game will increase/decrease depending on if you pressed the button when the green light was lit, the score counter will increase/decrease accordingly, and then the next level of the game on the new difficulty will begin, which means the main loop will begin again, with a different difficulty. The difficulty will increase by a constant amount each time, as we lack the ability to halve the time that each light stays on, without implementing a complex division algorithm in ARM assembly language.

Every LED, and the button, has to be connected to a ground pin. We connected a column on the breadboard to a ground pin, and then connected every LED to this column. If we used a separate ground for each LED/button, we would quickly realise we do not have enough ground pins.

The score counter will be very similar to the LED system for the main game, but will actually be a lot simpler, and will check to see if the register holding the player's score is a certain number, and then light up the LEDs accordingly. This will work like a giant switch statement, but in ARM assembly language.

Problems

When we were first implementing our extension, and came to using Assembler to compile our ARM assembly code for the first time, we found that our Assembler program had some bugs in it that we had not previously found. Luckily enough, we implemented lots of error messages to be printed to stdout when errors occur, so we could quickly find and fix these errors. The main error was that when using `strtol` to turn a string of a hexadecimal number into an int, it was converting it to a signed int, when we wanted it to be unsigned, rather than the hexadecimal constant being in two's complement representation.

Furthermore, we found that our ARM assembly code was extremely hard to understand after writing it, and was nearly unreadable, because there were so many lines of code, and we could not put any comments in the original assembly source file, as we had not allowed for this in the assembler. To fix this, we decided to add the functionality of comments into the assembler, so that we could all read through and understand the code we had written. Without these comments, only the person writing the code would know what was going on.

Before this project, nobody in our group really had any knowledge of how circuits and hardware worked. One of our members has a GCSE in electronics, which gave him some basic knowledge of how the different components work. We all have experience in the assembly language, as we all took the Architecture course, but we had not taken the Hardware course, as we are all JMC students. This meant that we had to do a lot of research on how circuits and electronics worked, and we had to spend a lot of time playing around with different circuits on the Pi, so that we got a better idea of how it all worked.

A cause of many problems in this project was the fact that we could not directly debug the ARM assembly code with a debugger like DDD or GDB. The best we could do was debug the assembler, with the assembly code as an argument to assemble. This did not always help as we could not see anything related to GPIO this way, for example. The only other thing we could do was heavily comment our ARM assembly code, and have everybody read through the code to look for errors.

The lack of a stack implementation meant that a lot of code had to be repeated, with the name of any labels slightly changed so that no two labels had the same name. This was painfully clear when programming the lines of code that make the program wait for a few seconds. We had to copy and paste this code every time we wanted the program to wait, and we had to slightly change the name of the labels to things like `wait1_1`, `wait1_2`, `wait2_1`, `wait2_2`, etc. If a stack was implemented, then we could have a wait method, and push parameters before calling the same method every time we want the program to wait.

Testing

We tested the ARM assembly program by running it in our assembler program, which was not entirely effective, as mentioned earlier, but definitely helped. We could use DDD to test the assembly code when it is an argument to the assembler, also as mentioned earlier. It would have been easier to debug the ARM assembly code on DDD directly, but we could not get this working. We tested the compiled binary file on our emulator before putting the program onto our Pi, as this allowed us to see if it would work on the Pi before loading the program onto the Pi. When writing the assembly code, we started off with simple assembly code, that would flash one LED for instance, and then gradually add more functionality to the program once we had debugged the previous code. This was a very good idea, as otherwise we would most likely be left with a very large ARM assembly program that did not work properly, which we could not directly debug with DDD or GDB, so we would not know which bit of the code was wrong, or if it was the set up of the circuitry that was wrong.

Group Reflection on Assembler & GPIO

After completing Emulator our team realised that we needed greater cohesion in deciding who is doing what and our overall understanding of the task. As we did not complete Emulator in time we decided to initially focus on Assembler and then return to Emulator once all our test cases for Assembler worked.

We began by having a discussion about what Assembler would entail. By the end of this conversation we had decided that our Assembler would consist of a main function that stored each line of code which were then turned to binary in an encode file. We also agreed about who would be focusing on which parts of the project (although we often ended up working together on specific areas). Furthermore we decided to communicate more over Facebook so that we would all remain up to date about what progress had been made and who might need help the following day (or things of that nature). Not only this but we also met every day in the labs just as with the emulator which meant that it was far easier to collaborate and get programs working.

Our style of organisation slightly changed for parts three and four, as they were smaller tasks than assembler we realised it would make sense if some of us focused on other parts of the project. For example, Kiran tidied our code for Emulator and Assembler whilst Rory focused on the Report. This worked well as Stefan had done electrical engineering at secondary school and Billy felt confident with assembly. As a group we feel that we made progress in terms of organisation; having a group discussion before beginning the task meant that we all knew what the task entailed and what we as individuals could do to help. This organisation meant that we finished assembler early and we had time to go back through Emulator and have

all the test cases working.

If we were to redo the task we might insist on a greater focus on tidy code. For example, magic numbers and uncommented code may have slowed down the progress of our project as it made understanding the code far more difficult. What we feel that we did well was our collaboration and communication. There was a great sense of openness in our group and a willingness to help each other. In addition, we were all very focused on completing the task with all members of the group showing great dedication.

Individual Reflections

Kiran

After Kiran finished the main file for Assembler with Billy he began to help Rory with SDT encode, where they spent much time trying to understand the strtok function. The particularly difficult part was figuring out a way to write a function that worked for post and pre indexed functions due to their different bracketing. When he finished SDT Kiran wrote many of the header files for the task, and then worked on the encoder for multiply functions. When Kiran first wrote his multiply function it did not compile,however, in the spirit of our team Billy went over it with Kiran until it passed the all of the tests. Kiran was also dedicated to debugging, spending almost two days optimising a loop test for Emulator so that our code passed all the tests on the test suite. One of Kiran's greatest strengths has been his enthusiasm for learning new things as the tasks developed, he also proved extremely hard working and patient. However, on occasion this may have acted as a weakness such as when he programmed at four in the morning, producing work that was not quite as good as his usual standard.

"Over the course of this project there have been many different challenges I have come across, however, with the help of my team I was always able to overcome them. For instance I was assigned the job of being the group leader; I feel that now looking back over the project as a whole, I should have taken some action to initial get the group more organised. If I were to work with another group I believe that producing an initial project structure to give a rough guide of how we would split the emulator between members of the group would remove any confusion for who was doing what at the start. This aside, we managed to learn from this mistake in assembler (as spoken about in the group reflection part of this report) and I felt that I tried my best to keep the group Facebook chat updated with frequent comments outlining the current status of the project and making sure any issues were clarified.

Starting off the project, I had not had any experience with programming in c and did not feel confident enough to move straight into coding the emulator. This may have made my initial input into the group seem slightly less than others. As time progressed I found that I became more determined to not let this lack of experience draw me back my input into the group. I would stay longer hours to make sure I was able to keep up with my group, stick to deadlines and get the code we produced to the best state possible. Mistakes and awkward segmentation faults that I or anyone else came across became more appealing to me as they only helped me to learn more about the language I was coding in. I realised this was the most efficient way for me to become better with c programming. My ability and confidence to code in c thus, improved drastically as a result of this and I will take this more dynamic way of learning into consideration in group projects to come. Most importantly, I have been able to experience how fulfilling it is to work together with others in my group and improve/combine the code we produced separately to form even better solutions to the problems we faced."

Billy

Once he finished the main file Billy single-handedly finished the branch encoder at an impressive speed. This meant that he had time to help others finish their parts of the project. Billy's technical knowledge allowed him to give great insight into the problems with our code; he subsequently spent large portions of his time debugging. As mentioned earlier he helped Kiran with multiply, he also helped Stefan with the lsl function which proved to be one of the most difficult parts of the task. Billy also debugged almost all our code syntactically, saving the other members of the group a large amount of time. Not only this but Billy wrote large amounts of the code for parts three and four, dedicating days to researching how the raspberry pi works. Billy's greatest strengths lie in his coding proficiency and his willingness to help others. Billy was also extremely dedicated to the project, working on his code throughout the night even after we left the labs, and making sure to communicate with the rest of the group. If Billy had a weakness it would be that his enthusiasm for the project compelled him to write overly long messages to other members of the group in order to keep them updated about his progress.

"I feel like working in a group has been very successful for this project. Being able to talk to other members of the group while working on the project, both for help and to hear their ideas on a certain part of the project, was extremely helpful. This was particularly apparent at the start of each section, when I would not really have too much of an idea what to do, or my idea might not be right, and talking through the specification with everybody helped me understand what we are actually meant to do, because everyone could help me understand. I liked to make sure that everybody knew what they were working on at any given time, and knew what still had to be done. This meant I was constantly communicating with the group, both inside and outside of labs. I think these were my greatest strengths in this project, as well as the fact that I was always willing to do work. I think that one of my weaknesses is that I was not using GitLab very responsibly, and was mostly just pushing to the master branch whenever I did some work, unless it was just some messing around to test something out. This sometimes made it frustrating for other members when they tried to push their changes, as they had to merge with my changes to the master. Also, my poor grasp of the C language at the start of the project led to me implementing a lot of the emulator in quite an inefficient way. This was storing a binary number as an array of 0s and 1s, as I did not realise that the computer already knows the binary representation of any number you enter, and is able to change between hexadecimal, binary, and decimal whenever it wants. This led to Kiran having to spend a day and a half, near the end of the project, trying to change this to a more efficient implementation (Just using `uint32_t` instead of arrays). Fortunately, I think I understood the C language very well after the first week. The feedback I received from WebPA backs up what I think my main strengths and weaknesses are, as the feedback says that I communicate very well, try to help everybody, and keep everybody motivated."

Stefan

As with Emulator Stefan was one of the first to grapple with how to approach Assembler; he wrote the structure for the encoder which created a lot of clarity for other members of the group when they were working on different types of encode functions. After completing the structure, Stefan worked on Data Processing functions in encode. During his time he wrote the shift and rotate functions in assemblerUtils.c. Stefan's greatest strengths in the task were his insight in understanding the task early on and helping others understand what they need to do to complete the project. Stefan showed his dedication to the project by not only undertaking large portions of the code, but also by programming some of the more difficult parts of the task such as the Data Processing and the Logical Shift Left function which would prove fundamental in testing the code for the Raspberry Pi. Not only this but Stefan had experience with electrical engineering, this allowed us to complete parts three and four far faster than with could have without him.

"In reflection of my performance and the WebPA feedback, I believe that I played my part in the team,

producing some good code and communicating well with the rest of the group to update them on the task I was currently working on. My group felt that I got a good grasp on the task at hand early on and therefore through the initial commits that I made, helped others understand the task better. I feel that since starting the project, I have now fully understood the basics of git, often using branches and the git mergetool to combine code together. Something that I realised I was not doing nearly enough was commenting my code. Since working in a team, I have realised just how useful it is to comment your code, not only to the benefit of my team-mates but also understanding your own code, a few weeks after writing it. At the very least commenting at the top of every function to illustrate what the function does will save a lot of time in the long run. Furthermore, whilst working in a team, I have learnt the importance of designing the implementation on paper first, giving everyone an overview of how everything fits together before we all split up and implement tasks individually. I have enjoyed working in a group, and learnt a lot from reading other people's code too."

Rory

Rory largely worked in collaboration with others such as on SDT in Assembler with Kiran, where he used his understanding of SDT from Emulator to make progress with Kiran. Rory also worked in collaboration with Stefan on Data Processing, such as writing the pseudo code for rotate which was later implemented by Stefan. After the Assembler was finished Rory also contributed to the debugging of Emulator. As well as doing the parts of work he was assigned to do, Rory was always willing to help with anything that anybody had any problems with.

"I greatly enjoyed working with this group again (we also did a Maths project together). They were all extremely hard working and willing to work as a team; everyone came into labs almost every day for the duration of the project. My main weakness is my programming ability, I found C quite a confusing language so I would often work with other people. That being said I do feel as if I contributed my fair share. If I were to work again with a group of people I would do more preparation before we started the task, I had only briefly read over the specification before Emulator and as a result found it difficult when we started working on it as a group. The main point of feedback from the WebPA is that I do not communicate enough outside of labs so I will definitely make an effort to improve this in future tasks."