Description:

You will use the [CPUlator simulator (Links to an external site.)](https://cpulator.01xz.net/" \t "_blank) to create an ARM assembly program to interface with an emulated UART device. When you are at the simulator’s website, remember to **choose “ARMv7” for the architecture and choose “ARMv7 DE1-SoC” for the system**. [This link provides details about interfacing with the UART device (Links to an external site.)](http://www-ug.eecg.utoronto.ca/desl/nios_devices_SoC/dev_jtaguart.html); this device is shown on the right-hand side of the screen, just below the seven-segment displays. Note that the JTAG (Joint Test Action Group) UART (Universal Asynchronous Receiver-Transmitter) uses a base address of 0xFF201000.

You will create a short ARM assembly program to echo whatever is typed into the UART device. This echo will switch the capitalization of any letters typed into the device; e.g., an upper-case “A” will become a lower-case “a”, and vice versa. To complete this properly, your program must do the following:

1. First, read from the UART and determine if the *Read FIFO* has a valid character. If it does not, then continue polling with a loop to see when a valid character arrives. If the *Read FIFO* contains a valid character, then continue with the next step.
2. Create a loop that will iterate through all characters in the *Read FIFO*; use the “number of characters to read” field to determine how many times you will need to iterate through the loop. Your loop should stop once you have iterated through all characters in the *Read FIFO*; to do this, branch to a different location in your code where you have a nop or a return.
3. Within this loop, echo back the characters to the *Write FIFO*, using the same base address, and switch the capitalization of any letters. Non-letter characters should be echoed, but not modified. I recommend that you examine an ASCII table when determining how to code this.

Recommendations:

* You may hard-code or use global variables/constants for common values that you may use repeatedly your program; e.g., base addresses or mask values.
  + Note, for a global variable or constant, you could just use a register that you use nowhere else in your program to hold the value
* I highly suggest that your approach your program should be the following:
  + First, design an algorithm using high-level language, pseudocode, flowcharts/diagrams, etc.
  + Next, translate this algorithm into ARM assembly then write instructions to test it

Successfully Completing this Job and Proving Correct Operation

To successfully complete this Job, you must do the following:

* Provide good comments for your ARM assembly
* Provide high-level language, pseudocode, flowcharts/diagrams, etc. you used to design your program in the report; **you must create and include this in your report for your design of this program**
* Your final assembly file must be **one file**

To prove the correct operation of your code, you need to test and take screenshots of the following:

* Show a **“before” screenshot** prior to stepping through your program once you have typed a very short message into the UART device. **Type no more than six characters** (the UART can handle more, but the *Read FIFO* text box is only large enough to show six characters) and **make sure to include** (1) at least one lower-case letter, (2) at least one UPPER-case letter, and (3) at least one non-letter character such as a space or punctuation character. Your screenshot should show:
  + The location of the next instruction to execute (where the PC register is pointing). This should be the first instruction of your program.
  + The ASCII characters of your short message in the *Read FIFO* of the UART device
* Show an **“after” screenshot** once you have stepped through the entire program and have reached the nop or return statement at the end of your program. Your screenshot should show:
  + The location of the next instruction to execute (where the PC register is pointing). This should be the nop or return statement at the end of your program.
  + The characters your program echoed in the text region of the UART device

Report:

Your report will usually be short, typically two to three pages (before adding your code or screenshots).

Your report should have the following sections:

1. **Introduction**
2. **Procedure/Discussion**
3. **Analysis**
4. **Conclusion**
5. **References**
6. **Appendix**

Please **use a title for each of these sections** like I have done in these instructions (this makes it easier for me to read) and **use complete sentences** in your report; **do not just write a bulleted list of items**. Please examine the parts of the rubric used for scoring each section.

Introduction: (See the conclusion section for scoring)

* Summarize the purpose of this job

Procedure/Discussion:

* Explain the design of your job. Provide your high-level language, pseudocode, flowcharts/diagrams, etc. you created for your design of your program.
* Share what challenges, difficulties, surprises (both good and bad), or ease you had when creating this program
* Provide “before” and “after” screenshots showing the correct operation of this program
* Describe if your program works correctly *(be truthful, I can tell from your code and screenshot(s) if it is correct or incorrect)*. If your program does not work correctly, then specifically describe:
  1. What is wrong?
  2. Where is the issue located in your code?
  3. How are you able to determine this?

Scoring for Procedure/Discussion (3 points possible)

* **Full credit (3/3) if:**
  + Explaining the design of the program:
    - Student thoroughly explains the process of writing the assembly program
    - Student includes high-level language, pseudocode, flowcharts/diagrams, etc. that were used to design the program
  + Proving correct operation of the program:
    - Student proves the code works properly or has a minor issue (but able to explain this thoroughly) with choice of test(s) and screenshot(s) and correctly identifies that the code works properly or has a specific, minor issue
* **1-point deduction for each of the following:**
  + Explaining the design of the program:
    - Student provides a good explanation of the process of writing the assembly program, *but did not provide information about an important part of the design process*
    - Student *forgets to include high-level language, pseudocode, flowcharts/diagrams, etc. that were used to design the program*
  + Proving correct operation of the program:
    - Student proves the code works properly or has a minor issue (but able to explain this thoroughly) with choice of test(s) and screenshot(s), *but does not identify that the code works properly or has a specific, minor issue*
    - Student *has some difficulty proving the code works properly (or has a minor issue) with choice of test(s) and screenshot(s)*, but correctly identifies that the code works properly or has a specific, minor issue
    - Student *proves the code has a major issue (but able to explain this thoroughly) with choice of test(s) and screenshot(s) and correctly identifies the specific, major issue with the code*
* **2-point deduction for each of the following:**
  + Explaining the design of the program:
    - Student provides a fair explanation of the process of writing the assembly program, *but did not provide information about multiple important parts of the design process*
  + Proving correct operation of the program:
    - Student proves the code has a major issue (but able to explain this thoroughly) with choice of test(s) and screenshot(s), *but does not identify the specific, major issue with the code*
    - Student *has significant difficulty proving the code works properly (or has a minor issue) with choice of test(s) and screenshot(s)*, but correctly identifies that the code works properly or has a specific, minor issue
    - Student *proves code is incomplete or has multiple major issues (but able to explain this thoroughly) with test(s) and screenshot(s) and correctly identifies the missing code and/or specific, major issues*
* **No credit (0/3) if:**
  + Explaining the design of the program:
    - Student explains the process of writing the assembly program, *but it is off topic or hard to follow*
  + Proving correct operation of the program:
    - Student *cannot prove the functionality of the code*
    - Student’s code is incomplete or has multiple major issues, *but does not identify the missing code and/or specific, major issues*
    - Student’s *code not applicable to this assignment*
  + Student *commits academic dishonesty*

Analysis:

Assume that you execute your code on the 5-stage pipeline (IF-ID-EXE-MEM-WB) that we’ve studied during lecture; this pipeline has one branch delay slot (the branch is evaluated in the ID stage). **Carefully look at the loop you created in your program to process the characters (do not examine any loop used for polling)** and please discuss the following calculations in your report:

* **How many clock cycles** are required to complete one iteration of your loop?
  + **To receive credit for this calculation, you must list your code and add any required stalls (similar to the ILP lecture); if you wish, you can instead include a pipeline diagram with each instruction labeled in your report.**
  + Consider control and data hazards when counting the number of clock cycles required
  + Assume that you are processing a **lower-case character** and each branch will cause one stall; e.g., branch delay slots are filled with a NOP
  + **Important:** Count the number of clock cycles from the IF stage of your first instruction of the first iteration of your loop up to, but not including, the IF stage of the second iteration of your loop
* Now, **unroll your loop once** so that your loop will handle two characters during each iteration. Additionally, **rearrange your code to eliminate as many stalls as possible and obtain the shortest number of clock cycles possible**. Determine **how many clock cycles are required to complete one iteration of your unrolled loop**, and the **speedup compared to your original loop**.
  + **To receive credit for this calculation, you must list your code and add any required stalls (similar to the ILP lecture); if you wish, you can instead include a pipeline diagram with each instruction labeled in your report.**
  + Consider control and data hazards when counting the number of clock cycles required
  + Assume that you are processing two **lower-case characters** and each branch will cause one stall; e.g., branch delay slots are filled with a NOP
  + **Again:** Count the number of clock cycles from the IF stage of your first instruction of the first iteration of your unrolled loop up to, but not including, the IF stage of the second iteration of your unrolled loop

Scoring for Analysis of Clock Cycle Count for Initial Loop (3 points possible)

* **Full credit (3/3) if:**
  + Student correctly determines the clock cycle count, or determines this with a very minor error, and provides either a list of the code with required stalls properly placed or an accurate pipeline diagram
* **1-point deduction for each of the following:**
  + Student determines the clock cycle count and provides either a list of the code with required stalls placed or a pipeline diagram, *but there are noticeable errors with one or two stalls*
  + Student determines the clock cycle count and provides either a list of the code with required stalls placed or a pipeline diagram, *but the clock cycle count does not seem to follow the correct list of code or pipeline diagram*
* **2-point deduction for each of the following:**
  + Student determines the clock cycle count and provides either a list of the code with required stalls placed or a pipeline diagram, *but there are noticeable errors with three or four stalls*
  + Student determines the clock cycle count and provides either a list of the code with required stalls placed or a pipeline diagram, *but the clock cycle count does not seem to follow the list of code or pipeline diagram with one or two stall errors*
* **No credit (0/3) if:**
  + Student determines the clock cycle count and provides either a list of the code with required stalls placed or a pipeline diagram, *but there are noticeable errors with five or more stalls*
  + Student determines the clock cycle count and provides either a list of the code with required stalls placed or a pipeline diagram, *but the clock cycle count does not seem to follow the list of code or pipeline diagram with three or four stall errors*
  + Student’s *work is incomplete*
  + Student *commits academic dishonesty*

Scoring for Analysis of Clock Cycle Count and Speedup for Unrolled Loop (3 points possible)

* **Full credit (3/3) if:**
  + Student correctly determines the clock cycle count and speedup, or determines these with a very minor error, and provides either a list of the code with required stalls properly placed or an accurate pipeline diagram
* **1-point deduction for each of the following:**
  + Student determines the clock cycle count and speedup and provides either a list of the code with required stalls placed or a pipeline diagram, *but there are noticeable errors with one or two stalls*
  + Student determines the clock cycle count and speedup and provides either a list of the code with required stalls placed or a pipeline diagram, *but the clock cycle count and/or speedup does not seem to follow the correct list of code or pipeline diagram*
* **2-point deduction for each of the following:**
  + Student determines the clock cycle count and speedup and provides either a list of the code with required stalls placed or a pipeline diagram, *but there are noticeable errors with three or four stalls*
  + Student determines the clock cycle count and speedup and provides either a list of the code with required stalls placed or a pipeline diagram, *but the clock cycle count and/or speedup does not seem to follow the list of code or pipeline diagram with one or two stall errors*
* **No credit (0/3) if:**
  + Student determines the clock cycle count and speedup and provides either a list of the code with required stalls placed or a pipeline diagram, *but there are noticeable errors with five or more stalls*
  + Student determines the clock cycle count and speedup and provides either a list of the code with required stalls placed or a pipeline diagram, *but the clock cycle count and/or speedup does not seem to follow the list of code or pipeline diagram with three or four stall errors*
  + Student’s *work is incomplete*
  + Student *commits academic dishonesty*

Conclusion:

* What did you learn or discover as you completed this assignment? Why is this important to microprocessor/microcomputer architecture?
* If you were to redo this assignment, what would you do differently? Why?
  + *If you would make no changes to how you would complete this assignment, then answer “No” and explain why*
* Do you have any questions after completing this assignment? *(Optional question, respond if needed)*
* Do you have any comments/suggestions for this assignment? *(Optional question, respond if you wish)*

References:

Include a list of references where you list any sources/websites used or students with whom you discussed this assignment. Use citations, use a citation style such as IEEE, APA, etc., in your report to show where you used the information from each source/website.

Appendix:

Include all your code (**copy-paste this as text**) for this assignment. **Note: You will not receive credit for this assignment if you do not include your code, or if you copy-paste this as an image.**

Scoring for Introduction and Conclusion (Your reflections about the assignment) (3 points possible)

* **Full credit (3/3) if:**
  + Student thoroughly introduces and reflects about the assignment
* **1-point deduction if:**
  + Student provides a good introduction and reflection about the assignment, *but forgot to introduce an important part of the assignment and/or forgot a response to one of the conclusion questions*
* **2-point deduction if:**
  + Student provides a fair introduction and reflection about the assignment, *but forgot to introduce multiple important parts of the assignment and/or forgot responses to multiple conclusion questions*
* **No credit (0/3) if:**
  + Student provides an introduction and/or conclusion, *but it is off topic or does not substantially discuss the assignment*
  + Student *commits academic dishonesty*