**CS2100 Tips**

**Exam Tips**

1. Do the **mechanical questions** first!

* For block diagram-related question, leave it to the end first
* Kmap too, but still try to allocate some time to attempt it if possible

1. Read the question **carefully** before you attempt it!

* It’s better you avoid careless mistakes rather than rushing to do the questions

1. Don’t straightaway do the question! Analyse the questions first to figure out the best way to approach the problem to save your time!

* This is esp true when analyzing MIPS code and dealing with questions related to circuits!

1. Be **meticulous** when solving the problems!

* i.e. be careful, don’t make careless mistakes!

1. For block level design question, a lot of times you have to draw out the **truth table** and try to find **patterns** from it
2. Remember that you can make use of the **complemented outputs (Q’)** of the **flip-flops!!!!!**
3. When filling up truth tables for flip-flops, remember to use the excitation table!

* Esp SR and JK flip-flops so that you can introduce more Don’t-cares!!!

1. There’s a **hack** to help you fill up the truth table for flip-flops faster using excitation table

* E.g. Use JK flip flop as an example, if we want to move from 0 to 1, then we just have to make sure that the set (J) is 1. Same thing, if we want to make sure that 0 remains in 0, we just have to make sure that the set(J) is **not 1,** i.e always 0.
* Use this concept and it should be able to help you to fill up the truth table faster

1. When filling up k-map, a way to help you to identify all the prime implicants is to try to think whether it is possible for you to find a larger grouping every time after you circle a group. See whether you can find an even larger grouping by **MOVING the grouping up, down, left, right, WRAP around, or USING the don’t-cares**

**Notes**

1. To write instructions in MIPS, first write out the instructions in C language first, then translate it to MIPS

* This will make your thought process much easier

1. Do the necessary **variables mapping first, including temporary variables that you will be using in your assembly code**

* Do this by analyzing the C Code and understand what are the registers that you need
  + E.g. a = (b + c) – (d – e)
  + Here, you may need two registers to store the temporary result of b + c and d – e respectively and then use the two registers to perform subtraction and assign the result to a.
* This will help in converting C code into MIPS code

**Instruction Set Architecture (ISA) Questions**

1. If there are more than 2 types of instructions and you are asked to minimize the number of instructions, use the extra bits of the opcode of each instruction as type identifier for the instructions with longer opcode (i.e. to differentiate between one the type of instructions with longer opcode)

* Refer to jotted note in notability for CS2100 Past Year Midterm 17/18 Sem 2

**Boolean Algebra**

1. In writing out terms, you should write the **literals in the order of significance**, especially in your final answer. For instance, for a Boolean function *F*(*j*, *k*, *m*, *p*), you should write the final answer as ***j*.*k'* + *j'*.p***'* and not *k'*.*j* + *j'*.*p'* or *j*.*k'* + *p'*.*j'.*

**Pipelining**

1. Tips in determining how many cycles are needed to execute a piece of code

* If there is **no data forwarding,**
  + Then the **ID** stage of an instruction, which has data dependency (register) on other instruction x, can only happen at the **WB** stage of the instruction X. Cuz the writing during WB happens during the first half of the cycle whereas the reading during ID only happen during the second half of the cycle.
  + Be careful of lw as well cuz it can incur delay
* If there is **no early branching**,
  + Then the **IF** stage of the instruction that immediately follows the branching instruction can only happen during the **WB** stage of the branching instruction because the branching decision is only computed during the MEM stage and can only be executed during the WB stage.
  + i.e. there will be **3 clock cycles delay**
* If there **is data forwarding**
  + Then be careful of lw cuz it usually will incur delay
  + Sw also
* If there **is early branching,** 
  + Then the **IF** stage of the instruction that immediately follows the branching instruction will happen at the **EXE** stage of the branching instruction because the branching decision is only computed at the ID stage and can only be executed at the next stage
  + i.e. there is still **1 clock cycle delay**.

1. Note that the data from the **memory** is only available **after** the MEM stage, i.e. at the start of the WB stage
2. To calculate the number of cycles executed in a MIPS code involving loops:

* **A:** First identify how many cycles are **not** included in the loop **before** the loop is executed (if any).
  + Be careful when doing this to make sure that the cycles in the loop are identical.
* **B:** Then identify how many cycles are involved in one loop
  + Be careful with this to ensure there is no overlapping between cycles of different loops (e.g. in tutorial 10q2d, the last W is not included as it is covered by the first F of the next loop / next instruction)
* **C:** Then identify how many cycles are **not** included in the loop **after** the loop is executed (if any).
* Then, the total number of cycles would be A + (B \* N) + C, where N is the number of iterations.
* Referring to the answer slides provided for tutorial 10 q2d, there are a total of 2 cycles that are **not** included in the loop **before** the loop (the IF and IF of the first two instructions)
* Then in each loop there are 18 cycles, note that the last W is not included as it is covered by the first F of the next loop / next instruction
* Then after the loop there are a total of 9 cycles not included in the loop (bne & beq)
* Hence the total number of cycles are = 2 + (4\*18) + 9 = 83