UNIVERSITY of York

Department of Electronic Engineering

Assessments 2019/20

ELE00067M

Digital Design

This assessment (**Homework 2**) contributes **20%** of the assessment for this module.

Clearly indicate your **Exam Number** on every separate piece of work submitted.

Unless the assessment specifies a group submission, you should assume all submissions are individual and therefore should be your own work.

All assessment submissions are subject to the Department's policy on plagiarism and, wherever possible, will be checked by the Department using Turnitin software.

Submission is via VLE and is due by **12:00** on **6 January 2020 (Spring Term, Week 1, Monday)**. Please try and submit early as any late submissions will be penalised.

Please remember that if this is your first year of study, you need to complete the mandatory Academic Integrity Tutorial http://www.york.ac.uk/integrity/

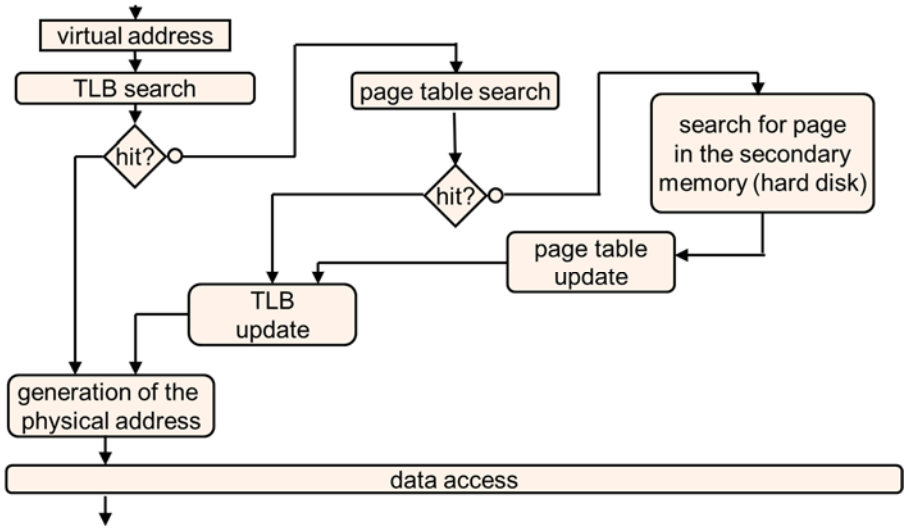
ELE00067M Digital Design - Homework Set 2

Try to answer all questions. This is an INDIVIDUAL effort. Access any material you deem useful (lecture notes, books, internet). You are allowed to discuss general approaches (for example, using the lecture slides as support) but not specific solutions, as this would be considered collusion.

IMPORTANT: This assessment should be submitted as a single PDF file through the VLE. Answer each question on a separate page. Write your exam number on every page. Failure to do so might result in your homework not being marked.

Question 1 *12 marks*

The diagram below, taken from the slides of lecture 10, illustrates the algorithm that controls the operation of pagination. Draw a similar algorithm for paged segmentation, assuming the presence of separate translation lookaside buffers (TLBs) for pages and segments. [12 marks]



Question 2 *12 marks*

Consider a processor with a 16KB cache organized in blocks of 64 32-bit words (256 bytes). Assume that addresses are coded in 32 bits and that there are 4GB of RAM without virtual memory (i.e., the 32 bit addresses in a program code directly the physical address in main memory, without any Memory Management Unit). All memories are byte-addressable. A complete block is retrieved from memory on a cache miss.
A program reads three one-dimensional arrays A, B, and C of 256 words each, one word at a time (that is, it reads in sequence A[0], B[0], C[0], A[1], B[1], C[1], A[2] … C[254], A[255], B[255], C[255]). Array A is stored in consecutive memory locations starting at hexadecimal address A1A10600, array B is stored in consecutive memory locations starting at hexadecimal address A1A20600, and array C is stored in consecutive memory locations starting at hexadecimal address BCADA600.

- For a direct-mapped cache, compute:
  - The number of bits required for the tag and for the index [1 mark].
  - The number of cache hits and misses as the program completely reads the three arrays [2 marks].
- For a fully-associative cache, assuming a least-recently-used replacement policy, compute:
  - The number of bits required for the tag (or address) [1 mark].
  - The number of cache hits and misses as the program completely reads the three arrays [2 marks].
- For a set-associative cache with 2 blocks per set, assuming a not-last-used replacement policy, compute:
  - The number of bits required for the tag and for the index [1 mark].
  - The number of cache hits and misses as the program completely reads the three arrays [2 marks].
- For a set-associative cache with 8 blocks per set, assuming a least-recently-used replacement policy, compute:
  - The number of bits required for the tag and for the index [1 mark].
  - The number of cache hits and misses as the program completely reads the three arrays [2 marks].

For all answers, show all the steps of the calculation and explain in a few words how you reached the results.

Question 3 — *26 marks*

Consider the following piece of code, which implements a multiplication algorithm:

```
01          loadi r2, 16
02          ori r30, r0, 4
03          loadr r3, r30
04          xori r29, r0, 65535
05          loado r4, r30, 3
06          move r7, r0
07          andi r5, r29, 16
08    L3:   andi r6, r4, 1
09          br r6, =0, +2 (L4)
10          add r7, r3, r7
11    L4:   shr r4, r4, 1
12          shl r3, r3, 1
13          dec r5, r5
14          br r5, ≠0, -6 (L3)
15          stori r7, 0
```

Identify all the data (RAW only) and control hazards in this code for the 5-stage pipeline of Architecture D (see lecture 8), described in the lectures and copied below for reference [4 marks].

To begin with, ignore the presence of any forwarding path (i.e., assume that there are no such paths) and assume that control hazards will be handled by the processor using pipeline stalls (i.e. no need to explicitly insert NOP instructions). Re-order the instructions to eliminate as many of the data hazards you have identified (without changing the function realized by the code, of course!). Then, insert the minimal number of NOP (no operation) instructions necessary to eliminate all remaining data dependencies. For your answer, rewrite the entire instructions and not just the line numbers. [7 marks].

Go back to the original code (without reordering and NOPs). Let F1 be the forwarding path from the Memory Access stage to the Execute stage, F2 the path from the Register Write stage to the Execute stage, and F3 the path between the Register Write stage and the Register Read stage (see figure below as a reminder). Identify which (if any) of the bypass paths can be used to eliminate each data hazard you have identified [3 marks].

The code above has two conditional branch instructions: B1 at position 09 and B2 at position 14. For a multiplication of 16 bit data, each of those branches is evaluated 16 times. B1 is taken or not depending on the value of the single bits of the multiplier, and hence it can be assumed that it will be taken, on average, 50% of the time. B2 is used to select which bit of the multiplier is evaluated, and hence will be always taken the first 15 times, not taken the last (16^{th}) time.

For Architecture D and ignoring data hazards (i.e., assuming that forwarding will solve any issues related to data hazards), compute how many clock cycles would be necessary to execute the above piece of code:

- Without any form of speculative execution [3 marks];
- With speculative execution using static prediction – predict not taken [3 marks];
- With speculative execution using static prediction – predict taken [3 marks];
- With speculative execution using static prediction – direction-based prediction [3 marks].

For all cases, show all the steps of the calculation and explain in a few words how you reached the results.