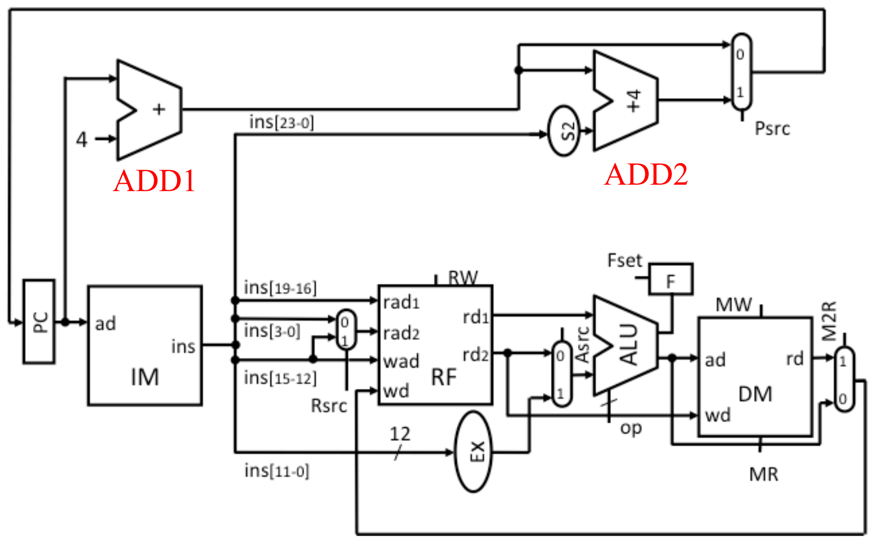
# COL 216 Computer Architecture
# Minor Test – II

Date: 24.03.2018 Time: 09:30 – 10:30 Max marks: 15

1. In the single cycle design shown here, the delays of IM, DM, RF and ALU are 200, 200,160 and 160, respectively (all in ps). ALU uses a full carry look-ahead adder. For the adders that do PC increment and offset addition to PC, there are choices of using carry propagate adder (delay = 600 ps) or carry look ahead adder (delay = 150 ps). It is well know that carry propagate adders are significantly cheaper than carry look ahead adder. What are the implications of different choices?



(3)

**Solution**:
The two adders are labeled as ADD1 and ADD2 in the figure. Different choices of components for these adders will influence delays of the paths passing through these. We look at various paths in two groups – (i) those which do not pass through the adders and (ii) those which pass through at least one adder. Largest of all the delays determines the maximum clock frequency.

(i) *Paths not passing through ADD1 or ADD2*

| Path | Instructions | Delay (in ps) |
|---|---|---|
| IM – RF – ALU – RF | DP | 200+160+160+160 = 680 |
| IM – RF – ALU – DM – RF | LDR | 200+160+160+200+160 = 880 |
| IM – RF – ALU – DM | STR | 200+160+160+200 = 720 |
| IM – RF – ALU | B | 200+160+160 = 520 |

[1 mark for computing the above delays, at least LDR path delay.]

(ii) *Paths passing through ADD1 and/or ADD2*

| | | Delays for different choices of (ADD1, ADD2) | | | |
|---|---|---|---|---|---|
| Path | Instructions | A.(cla, cla) | B.(cpa, cla) | C.(cla, cpa) | D.(cpa, cpa) |
| ADD1 | all | 150 | 600 | 150 | 600 |
| IM – ADD2 | B | 200 + 150 | 200 + 150 | 200 + 600 | 200 + 600 |
| ADD1 – ADD2 | B | 150 + 150 | 600 + 150 | 150 + 600 | 600 + 600 |
| | | | | | |
| Maximum delay over (i) and (ii) | | 880 | 880 | 880 | 1200 |
| Cost | | high | medium | medium | low |

[1 mark for computing delays for various adder choices].

Choice D (both adders as cpa) has the least cost but increases the maximum delay / clock period. Choice B or C (one adder as cpa and one as cla) results in medium cost without increasing max delay. Choice A (both adders as cla) has the maximum cost without any performance benefit over B and C.

[1 mark for overall conclusion.]

2. Consider a structural hazard free 5-stage pipelined design (IF-ID-EX-M-WB) for ARM instruction set {DP, DT, B} excluding auto increment/decrement, register offsets and shifts. The register file used here has 2 read ports and 1 write port. It is now proposed to enhance the design to include auto increment/decrement and register offsets for DT instructions, without modifying the register file. Would it introduce structural hazards? If yes, give illustrative examples for various cases and show the delays caused in each case.

(4)

**Solution**:
RF read/write requirements for all the variants of load/store instructions are tabulated below. The number of cycles is derived keeping in mind that RF has only 2 read ports and 1 write port.

✓

| LDR/STR instruction variants | Sources | Destinations | RD cycles | WR cycles |
|---|---|---|---|---|
| LDR with constant offset | Rn | Rd | 1 | 1 |
| LDR with constant offset, auto inc/dec | Rn | Rd, Rn | 1 | 2 |
| LDR with register offset | Rn, Rm | Rd | 1 | 1 |
| LDR with register offset, auto inc/dec | Rn, Rm | Rd, Rn | 1 | 2 |
| STR with constant offset | Rn, Rd | - | 1 | 0 |
| STR with constant offset, auto inc/dec | Rn, Rd | Rn | 1 | 1 |
| STR with register offset | Rn, Rd, Rm | - | 2 | 0 |
| STR with register offset, auto inc/dec | Rn, Rd, Rm | Rn | 2 | 1 |

Clearly, an LDR instruction with auto inc/dec requires an extra RF write access and an STR instruction with register offset requires an additional RF read access. Let us denote these instructions as LDR$^{auto}$ and STR$^{reg}$ for convenience. These instructions cause structural hazards. The exact details depend upon how the additional RF accesses are timed. Some examples are considered here.

Design 1: In one possible design, we assume 6-cycle timings for LDR$^{auto}$ and STR$^{reg}$ instructions, whereas other instructions have their usual timings, as shown below.

| | | | | | | |
|---|---|---|---|---|---|---|
| ✓ LDR$^{auto}$ | IF | ID | EX | M | WB | WB |
| STR$^{reg}$ | IF | ID | ID | EX | M | WB |
| DP | IF | ID | EX | M | WB | |

✓ Here instructions LDR$^{auto}$ and STR$^{reg}$ will have structural hazards with instructions that follow. Red ellipses show resource conflicts in the illustration below.

| | | | | | | |
|---|---|---|---|---|---|---|
| LDR$^{auto}$ | IF | ID | EX | M | WB | WB |
| DP | | IF | ID | EX | M | WB |

[1 mark]

| | | | | | | |
|---|---|---|---|---|---|---|
| STR$^{reg}$ | IF | ID | ID | EX | M | WB |
| DP | | IF | ID | EX | M | WB |

[1 mark]

This can be handled by introducing stall cycles (shown as □) as illustrated below.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| LDR$^{auto}$ | IF | ID | EX | M | WB | WB | | |
| DP | | IF | □ | ID | EX | M | WB | |
| DP | | | | IF | ID | EX | M | WB |

[1 mark]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| STR$^{reg}$ | IF | ID | ID | EX | M | WB | | |
| DP | | IF | □ | ID | EX | M | WB | |
| DP | | | | IF | ID | EX | M | WB |

[1 mark]

Design 2: In another possible design, we assume (all instructions follow 7-stage) timings as shown below.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| LDR$^{auto}$ | IF | ID | ID | EX | M | WB | WB |
| STR$^{reg}$ | IF | ID | ID | EX | M | WB | WB |
| DP | IF | ID | ID | EX | M | WB | WB |

The stages shown in red are dummy stages where the instruction doesn't use any resource. Here instructions LDR$^{auto}$ and STR$^{reg}$ will have structural hazards with preceding instructions, as illustrated below.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| DP | IF | ID | ID | EX | M | WB | WB | |
| LDR$^{auto}$ | | IF | ID | ID | EX | M | WB | WB |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| DP | IF | ID | ID | EX | M | WB | WB | |
| STR$^{reg}$ | | IF | ID | ID | EX | M | WB | WB |

This can be handled by introducing stall cycles as illustrated below.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| DP | IF | ID | ID | EX | M | WB | WB | | | |
| LDR$^{auto}$ | | IF | □ | ID | ID | EX | M | WB | WB | |
| DP | | | | IF | ID | ID | EX | M | WB | WB |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| DP | IF | ID | ID | EX | M | WB | WB | | | |
| STR$^{reg}$ | | IF | □ | ID | ID | EX | M | WB | WB | |
| DP | | | | IF | ID | ID | EX | M | WB | WB |

Design 3: In this design, we assume that the additional RF accesses required for LDR$^{auto}$ and STR$^{reg}$ instructions are not given separate cycles but combined with other cycles as shown below.

| | | | | | |
|---|---|---|---|---|---|
| LDR$^{auto}$ | IF | ID | EX | MW | WB |
| STR$^{reg}$ | IF | ID | XR | M | WB |
| DP | IF | ID | EX | M | WB |

Here MW means concurrent memory access plus RF write and XR means concurrent ALU operation plus RF read. In this case, instructions LDR$^{auto}$ will have structural hazard with preceding instructions and STR$^{reg}$ will have structural hazards with following instructions as illustrated below.

| | | | | | | |
|---|---|---|---|---|---|---|
| DP | IF | ID | EX | M | WB | |
| LDR$^{auto}$ | | IF | ID | EX | MW | WB |

| | | | | | | |
|---|---|---|---|---|---|---|
| STR$^{reg}$ | IF | ID | XR | M | WB | |
| DP | | IF | ID | EX | M | WB |

This can be handled by introducing stall cycles (shown as □) as illustrated below.

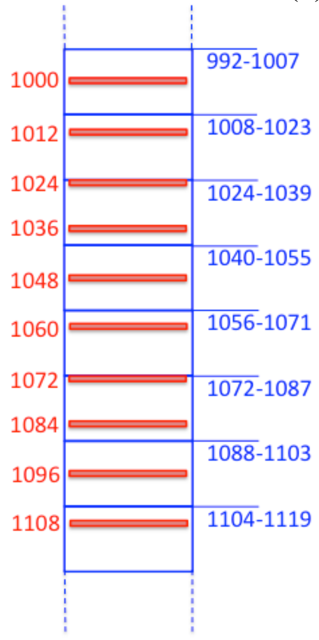
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| DP | IF | ID | EX | M | WB | | | |
| LDR$^{auto}$ | | IF | □ | ID | EX | MW | WB | |
| DP | | | | IF | ID | EX | M | WB |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| STR$^{reg}$ | IF | ID | XR | M | WB | | | |
| DP | | IF | □ | ID | EX | M | WB | |
| DP | | | | IF | ID | EX | M | WB |

3. A processor is connected to 32-bit wide memory through AHB-Lite bus supporting only single transfers and INCR4 burst transfers with SIZE parameter = 32. The bus data width is also 32-bits and the clock frequency is 50MHz. Memory has a latency of 90 ns, but after the first word, each subsequent word from the same 4-word block can be delivered with a latency of 15 ns (blocks start at byte addresses that are multiples of 16). Suppose it is required to read the words with addresses 1000 + 12*i where i = 0, 1, 2, … , 49. Suggest an efficient way of doing it. All numbers here are in base 10.

(4)

**Solution**:



Clock cycle time = 20 ns
For reading one word, duration of data phase = 90 ns = 5 cycles
Total time for read transaction (including address phase) = 6 cycles

A 4-word block can be read using INCR4 burst.
Latency for subsequent word reads from same block = 15 ns = 1 cycle
For reading one block, duration of data phase = 5+3 = 8 cycles
Total time for INCR4 burst (including address phase) = 9 cycles

The figure on right shows blocks with address ranges in blue and the words to be transferred in red. Two words that are within the same block can be read using INCR4 burst in 9 cycles, rather than two separate read word transactions that would take 12 cycles. Therefore, the best reading sequence is as follows.

Read word from address 1000
Read word from address 1012
Read block 1024-1039 (for words at addresses 1024 and 1036)

Read word from address 1048
Read word from address 1060
Read block 1072-1087 (for words at addresses 1072 and 1084)

and so on.

The pattern of 2 word reads followed by a block read will be repeated 12 times and there will be 2 words at the end, making a total of 26 word reads and 12 block reads. Total number of cycles for this = 26 * 6 + 12 * 9 = 156 + 108 = **264** [4 marks for this answer]. In comparison, if all words are read individually, then the number of cycle required is 50 * 6 = **300** [3 marks for this answer] and if the whole data is read in terms of blocks, there will be 38 block reads requiring 38 * 9 = **342** cycles [2 marks for this answer].
[1 mark if only no. of cycles for individual word read and INCR4 burst are correctly computed.]

In the above analysis, we have ignored pipelining and counted cycles for address phase for all transactions. If all transactions are pipelined, address cycle needs to be counted only for the first read. Accordingly, the above figures will change as follows.
264 => 1 + 26 * 5 + 12 * 8 = 227.
300 => 1 + 50 * 5 = 251.
342 => 1 + 38 * 8 = 305.

4. An application requires reading some text data from a device (e.g. a disk drive) and counting the number of occurrences of a string in that text. The data can be transferred from the device to the memory using DMA. The DMA controller can accept block size specifications of 512, 1024 or 2048 bytes. The amount of data (i.e., file size) is not known a priori, but is around a few tens of kilobytes. The device can indicate end-of-data to the DMA controller. Give the sequence of events from the point of view of the DMA controller for the entire process.

(4)

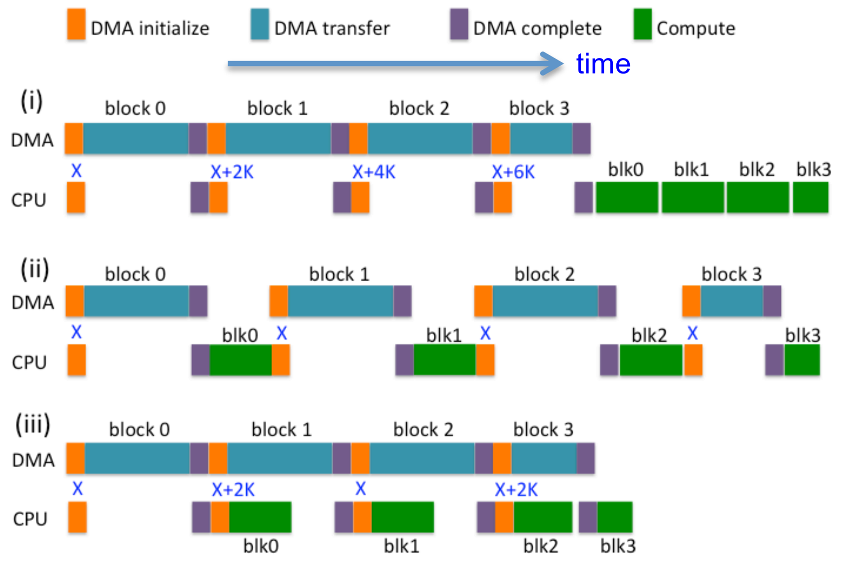
**Solution**:

The amount of data to be read could be much more than what can be handled in one DMA operation. Therefore it is clear that multiple DMA operations are required. Any choice of block size (512 / 1024 / 2048) is ok for this problem. In this solution we choose 2048.

The figure below illustrates a few possible schemes (a small value of text size = 7 K is considered here for illustration). It shows activities of the DMA controller as well as CPU. DMA initialization phase, DMA transfer phase and DMA completion phase and also the computation phase of CPU are shown in different colours.

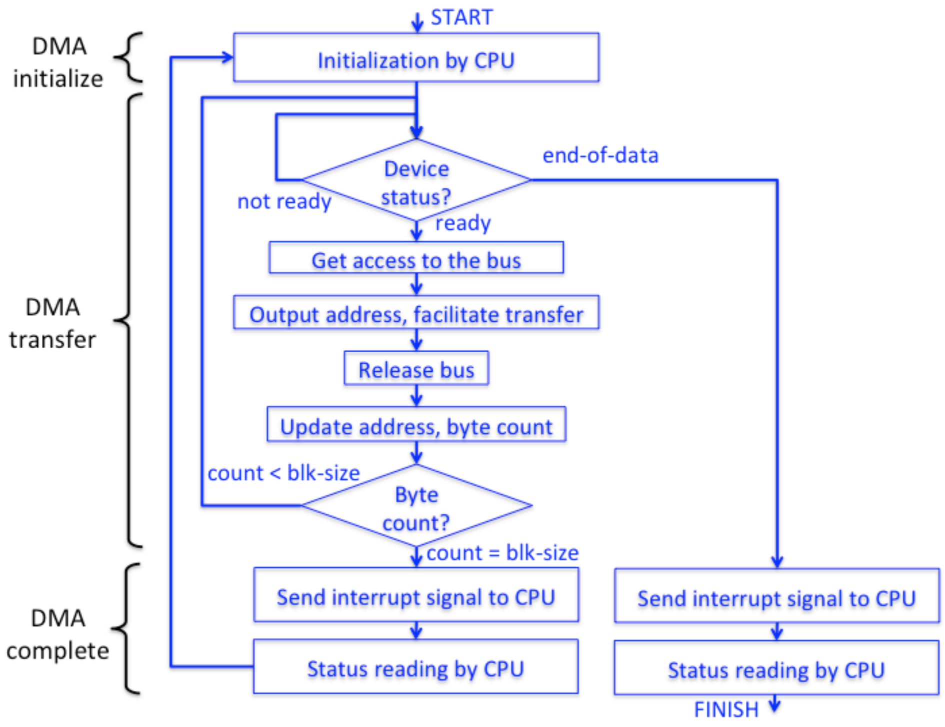
In scheme (i), the entire text is first read into memory and then the processing is carried out. Starting addresses of memory areas (X, X+2K, X+4K etc) are shown in blue. It is clearly wasteful in terms of memory space.

In scheme (ii) on the other hand, reading of a block is followed by processing it. Here we need space in memory just for one block. The starting address of memory is same for all operations and it need not be re-specified every time. In both these schemes, there is no concurrency between input/output and computation. Since the main purpose of using DMA is to free up CPU from the task of doing input/output so that it could do some computational work, these approaches are not efficient.



A good approach, shown as (iii), is to have computation and input/output to work in a pipelined fashion. We can have transfer of i+1th block to memory using DMA taking place concurrently with processing of ith block by CPU. This requires two buffers in memory (at addresses X and X+2Kin this illustration), each of size one block. When a block of data is being read into X, CPU operates on the previous block available at X+2K. After this the roles of the two buffers get reversed. The next block is read into X+2K and concurrently CPU works with X.

The sequence of events as seen by the DMA controller is shown in the form a flowchart given below.



In this flowchart, one iteration of the outer loop represents one DMA operation. One iteration of the inner loop represents transfer of one unit of data (byte/word etc.). It involves waiting for the device to be ready with the data, getting access to the bus, facilitating data transfer from device to appropriate memory address, releasing the bus, updating address and byte count and checking for end of operation. A DMA operation may end either on completing transfer of a block or getting an indication of end of data (e.g. end of file) from the device. In either case, an interrupt is sent to the CPU and appropriate status is passed on.

[The essential part of the answer is a flowchart or pseudo code or textual description covering the following events/operations.

|  |  |
|---|---|
| • DMA initialization | ½ mark |
| • Checking for device readiness / availability and end of data | ½ mark |
| • Bus access and release | ½ mark |
| • Keeping track of address and word/byte count | ½ mark |
| • Interrupt generation | ½ mark |
| • Giving status | ½ mark |
| • Iteration at byte/word level | ½ mark |
| • Iteration at block level | ½ mark |

1 bonus mark if overall scheme with concurrent I/O and computations is discussed.
]