CS425: Computer Systems Architecture

Homework Problem Set 2
Assignment Date: Friday 01/11/2024
Due Date: Wednesday 13/11/2024 23:59

Instructions: Solve all problems, create a .pdf file and send it via e-mail to HY425 course e-mail (hy425@csd.uoc.gr). Set the e-mail subject: HY425 - Homework 2

Problem 1 (50 points)

The following code is known as the DAXPY loop (**D**ouble-precision **AX** **P**lus **Y**) from the BLAS package (**B**asic **L**inear **A**lgebra **S**ubprograms), where **x** and **y** are arrays of doubles and **a** is a double:

```
for ( i=0 ; i<N ; i++ ){
     y[i] = a * x[i] + y[i];
}
```

Assume that our compiler has generated the following RISC assembly code:
[note: R1 keeps x[] index , R2 keeps y[] index, R4 keeps x[N-1] index, F0 keeps a]

Instruction	Notes
Loop: LD F2, 0(R1)	load x[i] into F2
MULTD F4, F2, F0	put a*x[i] into F4
LD F6, 0(R2)	load y[i] into F6
ADDD F6, F4, F6	put a*x[i] + y[i] into F6
SD F6, 0(R2)	store F6 into y[i]
ADDI R1, R1, #8	increment x index (R1)
ADDI R2, R2, #8	increment y index (R2)
SGT R3, R1, R4	test if loop done
BEQZ R3, Loop	loop if not done
NOP	branch delay slot

Further assume the following latencies of a typical 5-stage in-order fully-pipelined RISC processor (IF, ID, EX, MEM, WB) and that bypassing is applied whenever possible:

Operation(s)	Stage	Latency (cycles)
All Integer	EX	1
LD	MEM	2
SD	MEM	1
ADDD	EX	3
MULTD	EX	4

i. Show how the RISC processor would execute each loop iteration (indicate stalls) and calculate the total number of cycles required to run 1000 iterations of the loop.

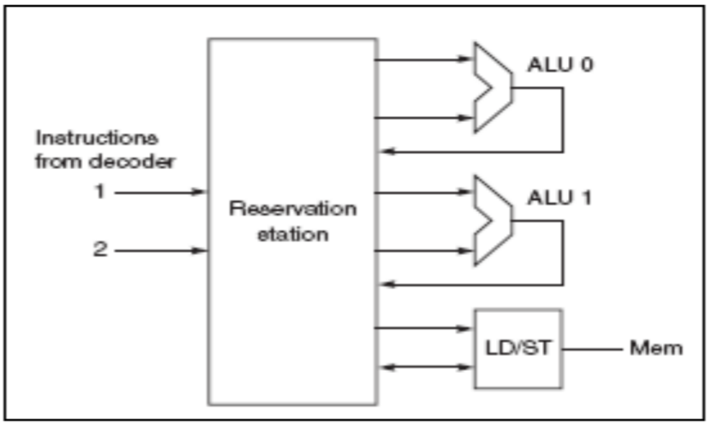
ii. Try to rearrange the instructions in order to reduce the number of stalls and then calculate the total number of cycles required to run 1000 iterations of the loop. Compare the performance with (i).

iii. Loop-unroll as many iterations needed, in order to reduce the number of stalls and then calculate the total number of cycles required to run 1000 iterations of the loop. Compare the performance now with (i) and (ii).

iv. Apply the technique of software pipelining and then calculate the total number of cycles required to run 1000 iterations of the loop. Compare the performance now with (i), (ii) and (iii). Do not forget the startup and cleanup code!

Problem 2 (50 points)

Let's consider the out-of-order microarchitecture shown in the figure below. Assume that we have a single the Reservation Station (RS) with "many slots" and that the ALUs can do all arithmetic ops (MULTD, DIVD, ADDD, ADDI, SUB) and branches. The RS can dispatch at most one operation to each functional unit per cycle (one op to each ALU plus one memory op to the LD/ST unit) – i.e. all functional units are pipelined and may complete successive instructions out-of-order.



i. Assume that all instructions from the **`Loop`** sequence provided below are already present in the RS (have been issued in-order), with no renaming having been done. Highlight any instructions in the code where register renaming would improve performance. Assume an infinite amount of registers and produce the register-renamed version of the code by using the following notation: the register F10 becomes F10a after the first renaming, F10a becomes F10b after the second renaming, etc. Hint: Look for RAW, WAR and WAW hazards. Assume the functional unit latencies on the table below.

```
Loop:      LD     F2,0(Rx)
I0:        DIVD   F8,F2,F0
I1:        MULTD  F2,F6,F2
I2:        LD     F4,0(Ry)
I3:        ADDD   F4,F0,F4
I4:        ADDD   F10,F8,F2
I5:        ADDI   Rx,Rx,#8
I6:        ADDI   Ry,Ry,#8
I7:        SD     F4, 0(Ry)
I8:        SD     F10,0(Rx)
I9:        SUB    R20,R4,Rx
I10:       BNZ    R20,Loop
```

Functional Unit Latencies	
Memory LD	2
Memory SD	1
Integer ADD, SUB	1
Branches	1
ADDD	3
MULTD	4
DIVD	6

ii. Assume that the complete register-renamed version of the code from part (i) is already present in the RS in clock cycle N (have been issued in-order and the values of the "ready" registers have been read) and assume the given functional unit latencies. Show how the RS should dispatch these instructions out-of-order, cycle-by-cycle, to obtain optimal performance on this code. Also assume that results must be written into the RS before they're available for use, i.e. no bypassing, and this takes 1 clock cycle. How many clock cycles does the code sequence take?

iii. Part (ii) allows the RS to optimally schedule these instructions. But in reality, the whole instruction sequence of interest is not usually present in the RS. Instead, various events clear the RS, and as a new code sequence streams in from the decoder, the RS must choose to dispatch what it has. Suppose that the RS is empty. In cycle 0 the first two register-renamed instructions of this sequence appear in the RS. Assume it takes 1 clock cycle to dispatch any op and assume the given functional unit latencies. Further assume that the front end (decoder/register-renamer) will continue to supply two new instructions per clock cycle. Show the cycle-by-cycle order of dispatch of the RS. How many clock cycles does this code sequence require now?