HY425

Homework Problem Set 2

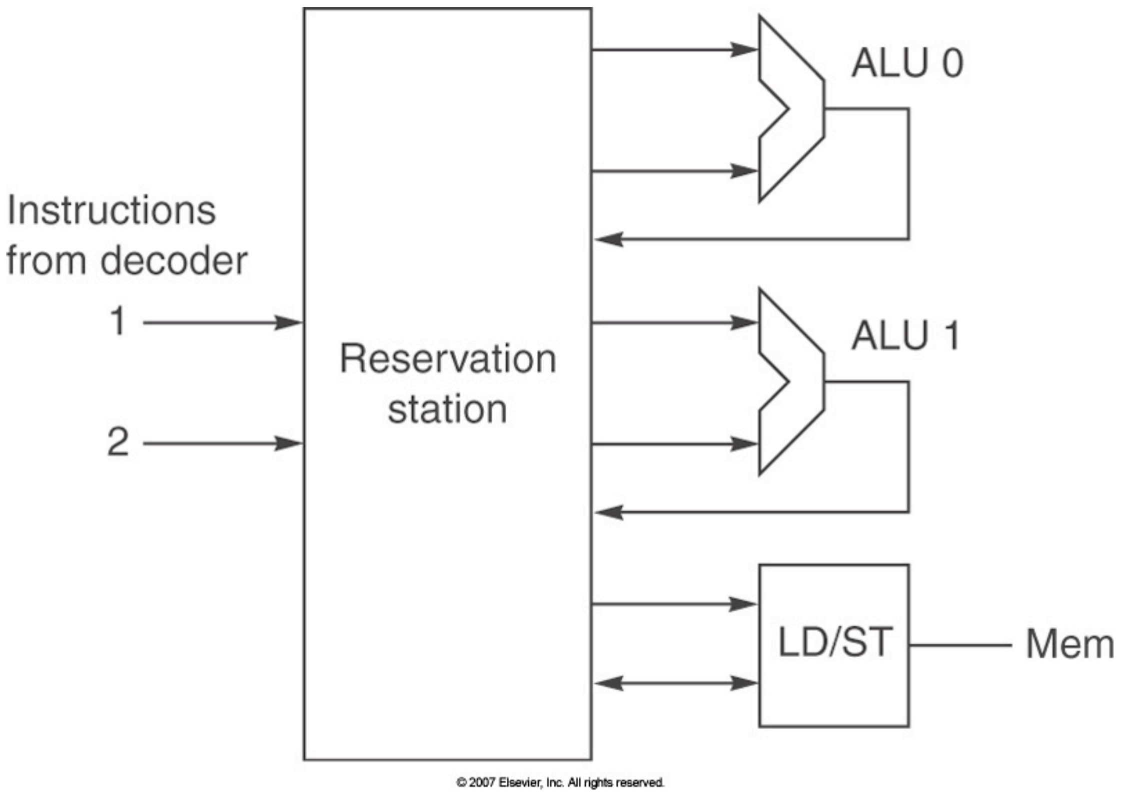
Assignment: 17/3/2008

Due Date: 31/3/2008

Instructions: Solve all problems in a **.pdf** file and send them via e-mail to Stamatis Kavadias (kavadias@ics.forth.gr), with a copy to the instructor (dsn@ics.forth.gr). Use the following subject in your e-mail: **HY425: Homework 2 Submission**. Please the aforementioned subject only, so that your homework is read and graded.

Problem 1 (50 points)

The following figure shows a microarchitecture with support for dynamic scheduling:


Instructions from decoder
1
2
Reservation station
ALU 0
ALU 1
LD/ST
Mem
© 2007 Elsevier, Inc. All rights reserved.

Assume that the ALUS can execute the arithmetic operations MULTD, DIVD, ADDD, ADDI, SUB and the branches, and that the Reservation Station (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). In other words, in each cycle, the architecture can start executing two ALU and one LD/ST operation. Assume the following code sequence:

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

The instructions have the following execution times:

Memory LD	4 cycles
Memory SD	2 cycles
Integer ADD, SUB	1 cycle
Branches	2 cycles
ADDD	3 cycles
MULTD	5 cycles
DIVD	11 cycles

The execution times include one cycle for dispatching the instruction from the RS to an execution unit, and the instruction execution latency. For example, the ADDD instruction takes 1+2 cycles to dispatch+execute.

a. Suppose all of the instructions from the sequence above are already present in the RS, i.e. their registers have been renamed appropriately. List again the instructions in the code (with their original register names) and highlight (e.g. circle, or mark in red) those instructions that can execute out-of-order and improve performance by overlapping otherwise idle cycles. To answer this question, you need to look for RAW and WAW hazards and find the instructions that can be reordered without violating dependencies. [9 points]
b. Suppose the register-renamed version of the code from part a. is again resident in the RS. Show how the RS should dispatch these instructions out-of-order, clock cycle by clock cycle, to obtain optimal performance. Assume the same RS restrictions and instruction latencies as in part a. Also assume that if two instructions are dependent, the first instruction must complete execution before the dependent instruction begins in the next clock cycle. In other words, there is no forwarding of values between dependent instructions, the source of the dependence must instead write its result first to the register file. How many clock cycles does the code sequence take in this case? [12 points]
c. In part b. we assumed that the whole instruction sequence is already in the RS, and the RS can apply dynamic scheduling to optimize performance. Unfortunately, it is not always possible to have all the instructions that we need to apply optimal dynamic scheduling available in the RS. As new instructions come in the RS from the instruction decoder, the RS must choose what instructions to dispatch for execution,

from the set of instructions available in the RS. Suppose that the RS is initially 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 instruction, and assume functional unit latencies are as they were for parts a. and b. Further assume that the decoder of the processor will keep supplying the RS with 2 instructions per clock cycle. Show the new cycle-by-cycle order of dispatch of instructions to the RS. How many cycles does this code sequence require now? It will be extremely helpful if you create a separate table showing cycle by cycle the instructions available for execution in the RS. [12 points]

d. If you wanted to improve the microarchitecture of part c. of this exercise which of the following options would have helped the most? A. another ALU, B. Another LD/ST unit, C. forwarding of ALU results to subsequent operations, more specifically, during the last cycle of execution of each ALU instruction forward the value to any dependent instructions, while simultaneously writing the result to the register file (this effectively reduces the latency between dependent instructions by one cycle), D. Cutting the longest instruction latency in half. What is the speedup attained in each case? [15points]

e. Let us assume that we can apply speculation in this microarchitecture. More specifically, assume that we can fetch, decode and execute instructions beyond one or more conditional branches. Suppose that you can speculate across one branch and fetch instructions from an additional loop iteration. Suppose also that you have all the instructions of two iterations readily available in the RS. How many clock cycles would the new microarchitecture need to execute the two loop iterations? [12 points]

Problem 2 (50 points)

The following C code implements a well-known computational pattern, called a "reduction". Assume that sum and A are integer variable and vector respectively and you are working with a 32-bit processor.

```
for (i=0;i<96;i++)

        sum += A[i];
```

You are given the following translation of the code:

```
        LA      R1, sum       # LA loads the address of a variable to a register
        LI      R2, 384       # LI loads an immediate constant to a register
        LA      R3, A         #assume R3 holds the memory location of "A[0]"
        LI      R4,0
Loop:   LW      R5,0(R3)
        ADD     R4,R4,R5      #sum += A[i]
        ADDI    R3,R3,4       #move pointer to A[i++]
        SUBI    R2,R2,4
        BNZ     R2,Loop
        SW      R4,0(R1)
```

a. Assume a standard pipelined processor where between a LW and a dependent arithmetic instruction, there is a load-use hazard, which causes the processor to stall for 2 cycles. How many clock cycles does the code require to execute, assuming that the processor provides forwarding in all other cases and the branch has a one-cycle delay slot? [6 points]

b. Your task is to optimize the code and eliminate stall cycles by using loop unrolling. Assume that you have 16 general purpose registers available on the processor (R1-R16). While thinking of the problem, you will discover that loop unrolling is not straightforward. You will first need to do a transformation of the original code so that you can apply unrolling effectively. It is helpful, albeit not required by the exercise, to think of the code transformation that you need to do to enable unrolling, in terms of the original C code. Show the unrolled code and count how many cycles the unrolled code requires, and what is the speedup over the original code of part a. [16 points]

c. Take a deep breath! Assume that you execute the same code on a VLIW machine where each long instruction can pack 2 integer arithmetic operations and/or branches, 2 memory (load/store) operations and these 4 operations can be dispatched simultaneously to different execution units. Using loop unrolling and based on the scheme you developed for part b. show the optimized VLIW code. How many cycles does the VLIW code require to execute the loop? What is the speedup compared to the original non-optimized code and compared to the optimized code in part b.? [22 points]

d. How many registers does your optimized VLIW code require? If each instruction of the code in parts a. and b. is 32-bits, whereas the VLIW instruction packet is 128 bits, calculate the VLIW “code blowup” factor. This is the ratio of the optimized code size in the VLIW microarchitecture to the non-optimized code size on the conventional pipelined microarchitecture. [6 points]