Computer Architecture and Engineering

CS152 Quiz #5

April 24th, 2008

Professor Krste Asanovic

Name:_____

This is a closed book, closed notes exam.

80 Minutes

10 Pages

Notes:

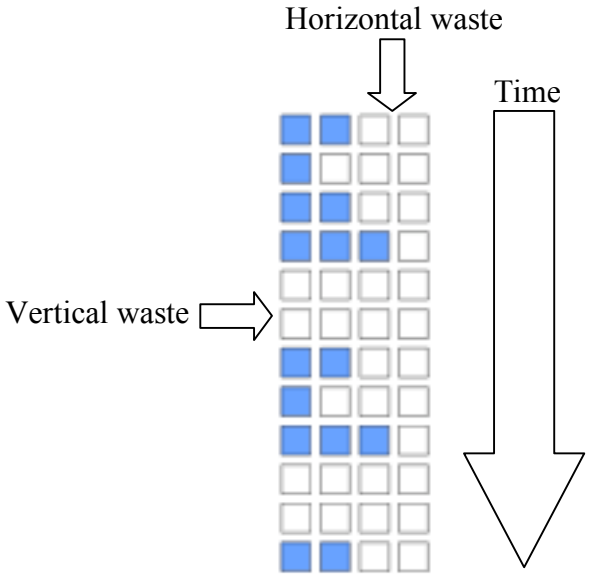
- **Not all questions are of equal difficulty, so look over the entire exam and budget your time carefully.**
- **Please carefully state any assumptions you make.**
- **Please write your name on every page in the quiz.**
- **You must not discuss a quiz's contents with students who have not yet taken the quiz. If you have inadvertently been exposed to the quiz prior to taking it, you must tell the instructor or TA.**
- **You will get no credit for selecting multiple choice answers without giving explanations if the instructions ask you to explain your choice.**

Writing name on each sheet	**_____**	**1 Point**
Question 1	**_____**	**32 Points**
Question 2	**_____**	**15 Points**
Question 3	**_____**	**17 Points**
Question 4	**_____**	**15 Points**
TOTAL	**_____**	**80 Points**

Problem Q5.1: General Comparison 32 Points

The goal of every one of the various architectures we have studied in this unit is to improve the utilization of the functional units built into the design. Achieving perfect saturation is often impossible, and in general we classify the wasted cycles as either vertical waste (due to long or variable latency instructions) or horizontal waste (due to limitations on the number of instructions that can issue or execute on a given cycle). Utilization is improved by exploiting parallelism, but the ways and times at which this parallelism is expressed vary radically between these architectures.

Fill out the taxonomy chart on the following page. You should fill in every box, even if the answer is "None" or "This is not done." All comparisons will be assumed to be relative to a simple in-order pipelined RISC machine unless you state otherwise.



	How is vertical waste caused by long latency instructions reduced?	How is horizontal waste caused by wide issue widths reduced?	Where and by what is most parallelism extracted (e.g., programmer, compiler, hardware)?	Name a limitation or disadvantage as compared to a simple in-order execution RISC machine
Out-of-Order Execution				
VLIW				
Vector				
Multithreading				
Simultaneous Multithreading				

Problem Q5.2: Predication and VLIW 15 Points

Consider the following code:

```
            l.s   f1, 0(r1)      ; f1 = *r1
            bneq  f1, f10, else  ; if f1==f10
            add.s f2, f1, f11    ;    f2 = f1 + f11
            b     if_end         ; else
else:       add.s f2, f1, f12    ;    f2 = f1 + f12
if_end:     s.s   f2, 0(r2)      ; *r2 = f2
```

Convert the code above to use predication rather than conditional branches. You should use the `CMPLTZ`, `CMPGEZ`, `CMPEQZ` or `CMPNEZ` instructions from Problem 5.3.B for predication, as reproduced below:

```
CMPLTZ pbit,reg     ; set pbit if reg < 0
CMPGEZ pbit,reg     ; set pbit if reg >= 0
CMPEQZ pbit,reg     ; set pbit if reg == 0
CMPNEZ pbit,reg     ; set pbit if reg != 0
```

You may use negative predication for instructions, e.g.:

```
(p1)  add r1, r2, r3    ; if (p1) r1 = r2 + r3
(!p1) add r1, r2, r3    ; if (!p1) r1 = r2 + r3
```

Predicated code for Q5.2:

Problem Q5.3: Multithreaded architectures 17 Points

The program we will use is listed below (in all questions, you should assume that arrays **A**, **B** and **C** do not overlap in memory.):

C code

```
for (i=0; i<328; i++) {
    A[i] = A[i] * B[i];
    C[i] = C[i] + A[i];
}
```

In this problem, we will analyze the performance of our program on a multi-threaded architecture. Our machine is a single-issue, in-order processor. It switches to a different thread every cycle using fixed round-robin scheduling. Each of the N threads executes one instruction every N cycles. We allocate the code to the threads such that every thread executes every Nth iteration of the original C code (each thread increments **i** by N).

Integer instructions take 1 cycle to execute, floating-point instructions take 4 cycles and memory instructions take 3 cycles. All execution units are fully pipelined. If an instruction cannot issue because its data is not yet available, it inserts a bubble into the pipeline, and retries after N cycles.

Below is our program in assembly code for this machine.

```
loop:  ld f1, 0(r1)     ; f1 = A[i]
       ld f2, 0(r2)     ; f2 = B[i]
       fmul f4, f2, f1  ; f4 = f1 * f2
       st f4, 0(r1)     ; A[i] = f4
       ld f3, 0(r3)     ; f3 = C[i]
       fadd f5, f4, f3  ; f5 = f4 + f3
       st f5, 0(r3)     ; C[i] = f5
       add r1, r1, 4    ; i++
       add r2, r2, 4
       add r3, r3, 4
       add r4, r4, -1
       bnez r4, loop    ; loop
```

Problem Q5.3.A **6 points**

What is the minimum number of threads this machine needs to remain fully utilized issuing an instruction every cycle for our program? Explain.

Problem Q5.3.B **3 points**

What will be the peak performance in flops/cycle for this program? Explain.

Problem Q5.3.C **8 points**

Could we reach peak performance running this program using fewer threads by rearranging the instructions? Explain.

Problem Q5.4: Vectorization 15 Points

State whether each of the following loops could be successfully vectorized and explain your answer. In all cases, you should assume that arrays **A**, **B**, **C** do not overlap in memory.

`for (i=0; i<N; i++)` `  B[i] = A[i] + C;`	
`for (i=1; i<N; i++)` `  B[i] = A[i] + B[i-1];`	
`for (i=0; i<N-1; i++)` `  B[i] = A[i] + B[i+1];`	
`for (i=0; i<N; i++)` `  C[i] = A[B[i]];`	

`for (i=0; i<N; i++)`
`  if(C[i] != 0)`
`  B[i] = A[i] + D;`	