

CS 222

Spring 2004

Final Exam **Name:**_____

Time: 3 hours

Instructions:

This exam consists of five questions for undergraduate students and six questions for graduate students. There are 13 or 15 pages with questions on them plus one additional pages at the end (p. 16) which is blank. Each of the five (or six) questions is worth the same weight. Please budget your time accordingly! Show all your work. Notes are NOT allowed. Calculators may be used if they have nothing useful in their memories.

1.a)

Consider the following code:

```
Loop: l.d    f1,0(r2)
      addi   r2,r2,#4
      l.d    f2,0(r2)
      add.d  f3,f2,f1
      mult.d f4,f3,f6
      s.d    f4, 0(r3)
      addi   r3,r3,#4
      addi   r4,r4,#-200
      bnez   r4,Loop
```

Show the timing diagram for one iteration of this instruction sequence for the classic five-stage RISC integer pipeline.

Assume there is normal forwarding hardware. Assume floating point add.d takes 4 cycles and floating point mult.d takes 7 cycles. Add extra columns to the timing diagram if needed.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
l.d f1,0(r2)																
addi r2,r2,#4																
l.d f2,0(r2)																
add.d f3,f2,f1																
mult.d f4,f3,f6																
s.d f4, 0(r3)																
addi r3,r3,#4																
addi r4,r4,#-200																
bnez r4,Loop																

1b) A common transformation required in cryptography is SQUARING. Implementations of squaring vary significantly in performance, especially among processors designed for encryption. Suppose SQUARING is responsible for 40% of the execution time of an encryption benchmark. One proposal is to enhance the SQUARING hardware and speed up this operation by a factor of 10. The other alternative is just to try to make all MULTIPLY operations in the encryption processor by a factor of 3; MULTIPLY instructions are responsible for a total of 60% of the execution time for the application. The design team believes that they can make all MULTIPLY instructions run 3 times faster with the same effort as required for the fast SQUARING. Compare the two design alternatives. [Note that SQUARING is a MULTIPLY operation.]

2. Consider the following loop.

```

Loop: l.d    f0,0(r1)
      add.d  f0,f0,f2
      l.d    f4,0(r2)
      add.d  f0,f0,f4
      add.d  f0,f0,f4
      s.d    f0, 0(r2)
      addi   r1,r1,#-8
      addi   r2,r2,#-8
      bnez   r1,Loop
  
```

and assume a single-issue pipeline with latencies as follows

instruction producing result	instruction using result	latency in clock cycles
FP ALU op	FP ALU op	3
FP ALU op	store double	2
load double	FP ALU op	1
load double	store double	0

As well, assume an integer load latency of 1 and an integer ALU operation latency of 0. Also, assume a one cycle delayed branch.

a) Without unrolling or scheduling, fill in the table below. Include any stalls or idle clock cycles. See the latency table above.

	clock cycle issued
l.d f0,0(r1)	1
add.d f0,f0,f2	
l.d f4,0(r2)	
add.d f0,f0,f4	
add.d f0,f0,f4	
s.d 0(r2),f0	
addi r1,r1,#-8	
addi r2,r2,#-8	
bnez r1,Loop	

2b) Assume there is one branch-delay slot, unroll and schedule the code so that it has no stall. What is the minimum number of iterations needed to reduce all stall?

```
Loop: l.d    f0,0(r1)
      add.d  f0,f0,f2
      l.d    f4,0(r2)
      add.d  f0,f0,f4
      add.d  f0,f0,f4
      s.d    f0,0(r2)
      addi   r1,r1,#-8
      addi   r2,r2,#-8
      bnez   r1,Loop
```

2c) Consider a loop:
for (i=1; i<=100; i=i+1) {
A[i]=A[i]+B[i]; /* S1 */
B[i+1]=C[i]+D[i]; /* S2 */
}

What are the dependences between S1 and S2? Is this loop parallel? If not, show how to make it parallel?

3. Consider the execution of the following loop. Use a MIPS pipeline extended with a two-issue Tomasulo's algorithm. Assume separate integer functional units for effective address calculation, for ALU operations, and for branch condition evaluation.

Fill in the tables below in parts a) and b).

```

Loop:  l.d    f2,0(r1)
       add.d  f4,f2,f0
       l.d    f6,0(r2)
       add.d  f6,f4,f6
       s.d    f6,0(r2)
       addi   r1,r1,#-8
       addi   r2,r2,#-8
       bnez  r1,Loop

```

Use a MIPS pipeline extended with a two-issue Tomasulo's algorithm. Assume that both one floating-point and one integer operation can be issued on every clock cycle, even if they are dependent.

Assume there are two integer functional units, one is used for ALU operations and the other is for effective address calculations. Assume that a separate pipelined FP functional unit for each operation type. Assume that there is dynamic branch prediction hardware and a separate functional unit to evaluate branch predictions/conditions. Assume that branches single issue. Assume there are two CDBs.

Assume that Issue and Write take 1 clock cycle. Integer operations take one cycle in their execution unit, and FP additions take 3 cycles in theirs.

iteration number		instructions	issues at	executes	memory access at	write CDB at
1	l.d	f2,0(r1)	1			
1	add.d	f4,f2,f0				
1	l.d	f6,0(r2)				
1	add.d	f6,f4,f6				
1	s.d	f6,0(r2)				
1	addi	r1,r1,#-8				
1	addi	r2,r2,#-8				
1	bnez	r1,Loop				
2	l.d	f2,0(r1)				
2	add.d	f4,f2,f0				
2	l.d	f6,0(r2)				
2	add.d	f6,f4,f6				
2	s.d	f6,0(r2)				
2	addi	r1,r1,#-8				
2	addi	r2,r2,#-8				
2	bnez	r1,Loop				

3b) Assume speculation and two-issue Assume that two instruction can commit per clock cycle. Fill in the table below. The first column in the table indicates what iteration of the loop the instruction is in. Unless specify, other assumptions are the same as part a).

it. no.	instr.	issues at clock cycle number	executes at clock cycle number	read access at clock cycle number	write CDB at clock cycle number	commits at clock cycle number
1	l.d f2,0(r1)	1				
1	add.d f4,f2,f0					
1	l.d f6,0(r2)					
1	add.d f6,f4,f6					
1	s.d f6,0(r2)					
1	addi r1,r1,#-8					
1	addi r2,r2,#-8					
1	bnez r1,Loop					
2	l.d f2,0(r1)					
2	add.d f4,f2,f0					
2	l.d f6,0(r2)					
2	add.d f6,f4,f6					
2	s.d f6,0(r2)					
2	addi r1,r1,#-8					
2	addi r2,r2,#-8					
2	bnez r1,Loop					

3c) Draw the transition diagram for the 2-bit prediction scheme.

4.

a) Briefly state and describe five techniques for reducing the miss penalty caches.

4b) Calculate the overall miss rate and AMAT for a 32kB instruction cache with a 32kB data cache. These are write through caches with a write buffer. Ignore stalls due to the write buffer. Assume that the instruction cache has 1.36 misses per 1000 instructions and the data cache has 38.4 misses per 1000 instructions. Also assume that 50% of all instructions are data transfer instructions. Assume a hit takes 1 clock cycle and the miss penalty is 80 clock cycles.

4c) Describe the following terms: Write back, write allocate.

5.

Suppose that in 1000 memory references there are 50 misses in first-level cache, 30 misses in the second-level cache and 20 misses in the third-level cache. Assume the hit time for the L1 cache is 1 clock cycle, the hit time for the second level cache is 10 clock cycles, the hit time for the third level cache is 50 clock cycles, and the miss penalty from the L3 cache to main memory is 100 clock cycles.

a) What is

(i) the local miss rate for L1?

(ii) the local miss rate for L2?

(iii) the local miss rate for L3?

(iii) the global miss rate for L3?

b) What is the average memory access time, AMAT?

c) Compiler Optimizations:

Describe one technique to use compiler optimizations to reduce miss rate.
Give a fragment of code to illustrate your idea.

6. Graduate student only.

a) Increasing the size of a branch-prediction buffer means that it is less likely that two branches in a program will share the same predictor. A single predictor predicting a single branch instruction is generally more accurate than is that same predictor serving more than one branch instruction.

i) List a sequence of branch taken and not taken actions to show a simple 1-bit predictor sharing that reduces misprediction rate.

ii) List a sequence of branch taken and not taken actions that show a simple example of how sharing 1-bit predictor increases misprediction.

b) In class, we have dealt mostly with the case when the number of iterations k we unroll is a factor of n (the total number of iterations.) [We assume that when $k = 1$, this corresponds to the original code without any unrolling for consistency purpose; when $k = 2$, we put 2 iterations.] Consider the case when k is not a divisor of n . The compiler would break n iterations into $\lfloor \frac{n}{k} \rfloor$ iterations of of k iterations together with $n \pmod k$ iterations of 1. [i.e. the n iterations is broken into 2 loops, one with $\lfloor \frac{n}{k} \rfloor$ and the other with $n \pmod k$ iterations.]. We then schedule the codes of each of the two loops to obtain the optimal performance.

Consider the following piece of code:

```

Loop:  l.d    f0,0(r1)
        add.d f4,f0,f2
        s.d   f4,0(r1)
        addi  r1,r1,#-8
        bne   r1,r2,Loop

```

Assuming the latencies as in Question 2, and assuming there is one branch-delay slot. Due to the code space restriction, k is bounded by at most 50. It is given that $r1 = r2 + 332$. What value of k would minimize the number of cycles needed for this code. What is the resulting code? How many instructions are there? How many cycles does it take to execute the code? Full logical reason is REQUIRED to obtain FULL credit.

—
End of exam

Have a great summer!

blank page