CS 352H Computer Systems Architecture
Exam #1 - Prof. Keckler
October 11, 2007

Name: ______**Solutions**________
(please print)

1-3.	11 points	________
4.	7 points	________
5.	7 points	________
6.	20 points	________
7.	30 points	________
8.	25 points	________
	Total (105 pts):	________

In recognition of University regulations regarding academic dishonesty, I certify that I have neither received nor given unpermitted aid on this examination. One sheet of notes and a non-programmable calculator are allowed.

Signed:___________________________

Multiple Choice

The following problems have one or more correct responses. Circle the letters corresponding to **ALL** of the correct answers. *(3 pts each)*

1. Which of the following statements about semiconductor technology trends are true?

   a. Transistors are getting faster because they are getting smaller.

   b. Because individual transistors are consuming less power, chip power consumption is decreasing.

   c. Smaller transistors are more reliable, meaning that they are less susceptible to cosmic rays.

   d. Chip bandwidth is not keeping up with the raw on-chip calculation capabilities.

   e. Computation is becoming cheaper while communication is becoming relatively more expensive.

2. Which of the following are true statements about control hazards?

   a. Control hazards are less challenging to minimize than data hazards.

   b. Delayed branches completely eliminate the performance impact of control hazards.

   c. The performance of a pipeline depends on the accuracy of its branch predictor.

   d. Hardware can do a better job predicting branches than a programmer or compiler.

Matching

3. Place the following mathematical functions in order of latency (cycles for the operation). Indicate the order (1-5, where 1 is the fastest) in the blanks below *(5 pts)*.

   _**2**_ Double-precision floating-point adder

   _**1**_ 32-bit carry-lookahead integer adder

   _**4**_ 64-bit integer multiplier

   _**3**_ Single-precision floating-point multiplier

   _**5**_ 32-bit integer divider

Short Answer

4. The authors of your textbook are both credited (along with John Cocke at IBM) with creating reduced instruction set computers. While simplicity of the architecture is attractive, explain why the x86, a complex instruction set architecture, has not been superseded by RISC architecture. *(7 pts)*

RISC machines offered promise of higher performance largely through pipelining and higher clock rates. Simple instructions are easier to pipeline as each instruction does a small amount of work, whereas in a CISC machine, some instructions are simple, while others may take a large number of cycles to complete. In addition, RISC machines typically have a fixed instruction format, which is easier to decode than a variable length format, often found in CISC machines.

Nonetheless, the x86 has thrived for a couple of reasons. First, it has a very large installed software base, meaning that people want to buy the machines because the software they want is readily available. Switching to a new platform may mean porting their programs, which is not always an easy task. Second, from a more technical perspective, Intel and AMD have figured out how to translate the complex instructions into simple RISC-like instructions on-the-fly. Thus the processor core can be similar to that of a RISC machine and benefit from pipelining and high clock rates.

5. Ideally, a processor with a 5 stage pipeline will be 5 times as fast as an unpipelined processor. Give at least two reasons why this ideal cannot be met and provide examples. *(7 pts)*

1) Control hazards: branch and jump instructions take a few cycles to compute their target addresses, which is too late to be used to fetch the first instruction after the branch in time. This is often solved by stalling the pipeline until the branch target has been resolved or by exposing branch delay slots in the instruction set architecture. A little later, we will learn about branch prediction as a means to further mitigate control hazards.

2) Data hazards: In the simple 5-stage pipeline, an instruction that needs the result of a load instruction may have to be stalled until the load completes. These read-after-write (RAW) hazards cause pipeline stalls and increase the average CPI. The same problem can occur with any multicycle instruction (such as a multiply). In more sophisticated out-of-order pipelines, some of these hazards can be masked by the hardware finding an alternative instruction to execute. We will cover this later as well.

3) Memory latencies larger than one cycle: In real life, one cannot access main memory (DRAM) in a single cycle. Instead, processors use a cache hierarchy to reduce average memory latency and in many processors, most of the load instructions can be serviced by the cache in a single cycle. When the data cannot be found in the cache, the pipeline may have to be stalled for a long period of time. We will learn more about cache hierarchies shortly.

4) Latch delay: Each clock cycle will not be 1/5 as long as the original cycle time since we need to budget some time for the pipeline latch. With faster clock rates from deeper pipelines, this relative overhead increases.

Other possibilities include structural hazards and delays to refill the pipeline.

Analytical Problems

6. As we discussed briefly in class, microprocessor designers have extended instruction sets to provide for explicit instruction-level parallelism. For example, in 1999 Intel introduced the Streaming SIMD Extensions (SSE; SIMD stands for single instruction, multiple data) which enabled a given instruction to operate on more than one piece of data. Others followed suit, such as Altivec for the PowerPC and MDMX for MIPS. Using these SIMD (aka vector) instructions can accelerate certain types of programs. Consider the C code below assembled into regular MIPS and MIPS with SIMD extensions. Note that the new vector instructions operate on 128 bits at a time: `lv128` loads 128 bits from memory and `vadd` adds four 32-bit numbers at a time. Note also that the "v" registers are 128 bits wide.

```
for(i=0; i<ARRAY_END; i++) {
  c[i] = a[i] + b[i];
}
```

```
;; Regular MIPS
LOOP:
 lw $t1, 0($a)
 lw $t2, 0($b)
 add $t1,$t1, $t2
 sw $t1, 0($c)
 add $a, $a, $4
 add $b, $b, $4
 add $c, $c, $4
 bne $a, $max_a, LOOP
```

```
;; MIPS with SIMD extensions
 sra $max_a, $max_a, 2
LOOP:
 lv128 $v1, 0($a)
 lv128 $v2, 0($b)
 vadd $v1,$v1, $v2
 sv128 $v1, 0($c)
 add $a, $a, $16
 add $b, $b, $16
 add $c, $c, $16
 bne $a, $max_a, LOOP
```

a. How many iterations of the loop are required when using SIMD extensions, relative to the original code? *(5 pts)*

b. Assume that a program spends 60% of its time in above loop on a machine without SIMD extensions. How much faster will the program with the SIMD? State any additional assumptions you need to solve the problem (but make your assumptions reasonable). *(10 pts)*

c. What might you do in the ISA to make this program go yet faster? What is the theoretical speedup limit that can be provided by the SIMD extensions for the program referred to in (b). *(5 pts)*

a) Since the vector length with the vector extensions is four 32-bit elements, the loop using the extensions only needs to execute one quarter as many times.

b) This is a classic Amdahl's law problem, which means to compute speedup, we need to figure out what fraction of the time of the original program can be sped up (p) and what factor we can speed it up by (S). Since 40% of the time is in code without the SIMD extensions, then p = 0.6. Part a indicates that since we can execute 1/4 of the iterations, then we should get a speedup of approximately a factor of 4. Thus Amdahl's law says that $T_{new}/T_{old} = (1-p) + p/S = 0.4 + 0.6/4 = 0.55$. Thus speedup=1/.55=1.82.

c) To get further speedup, one could make the vector length longer. In the limit where the vector length is infinite, $T_{new}/T_{old}=0.4$ and the speedup = 2.5. To do any better, one would have to find more places in the program to use the SIMD extensions.

7. In this problem, you will consider a new pipeline organization that eliminates the offset from the MIPS memory addressing mode. Thus an instruction like `lw $t1,4($t2)` would no longer exist, and the substitute would be two instructions: `add $t3,$t2,4` followed by `lw $t1,$t3`. The pipeline organization is shown below.

$$\begin{tikzcd}
\mathbf{F} \arrow[r, dash] & \mathbf{R} \arrow[r, dash] \arrow[dr, dash] & \mathbf{X} \arrow[r, dash] & \mathbf{W} \\
 & & \mathbf{M} \arrow[ur, dash] &
\end{tikzcd}$$

a. What advantage would this organization have over the standard five stage pipeline? How does it affect the hazards of the pipeline? *(5 pts)*

b. Assume the instruction mix below for the standard 5-stage pipeline. Now assume that 80% of the loads and stores in the original program for the original pipeline use a zero offset. What is the instruction mix for the new pipeline (fill in the table)? How many instructions must be executed for the new ISA and pipeline, compared to the old one? *(5 pts)*

Instruction Type	**Old Mix %**	**CPI**	**New Mix %**
ALU	40	1.0	45
load	30	1.6	27.4
store	15	1.0	13.8
branch	15	2.0	13.8

c. Based on the old CPI shown in the table above, what fraction of the load instructions cause a pipeline stall due to a data hazard? What is the average CPI of the old pipeline and what is the average CPI of the new pipeline? *(5 pts)*

d. Which machine is faster and by how much? Show your work and state any assumptions you need to make. *(10 pts)*

a) The potential advantage of this pipeline is that it completely eliminates the load delay slot. An instruction needing the result of a load can get it bypassed immediately without a bubble in the pipe. Of course, we'll need more instructions (see below).

b) With the ISA and pipeline modifications, we will have the same number of loads, stores, and branches as before, but we'll need more ALU operations to account for address calculations that are no longer done by the loads and stores. Imagine a program with 100 instructions and the mix above. Then there will be 45 loads and stores, of which 20% will need an ALU operation to compute the address. This leads to 9 additional ALU operations, or an increase in instruction count (IC) of 9%. The new instruction mix is shown above, accounting for a renormalization to the larger IC.

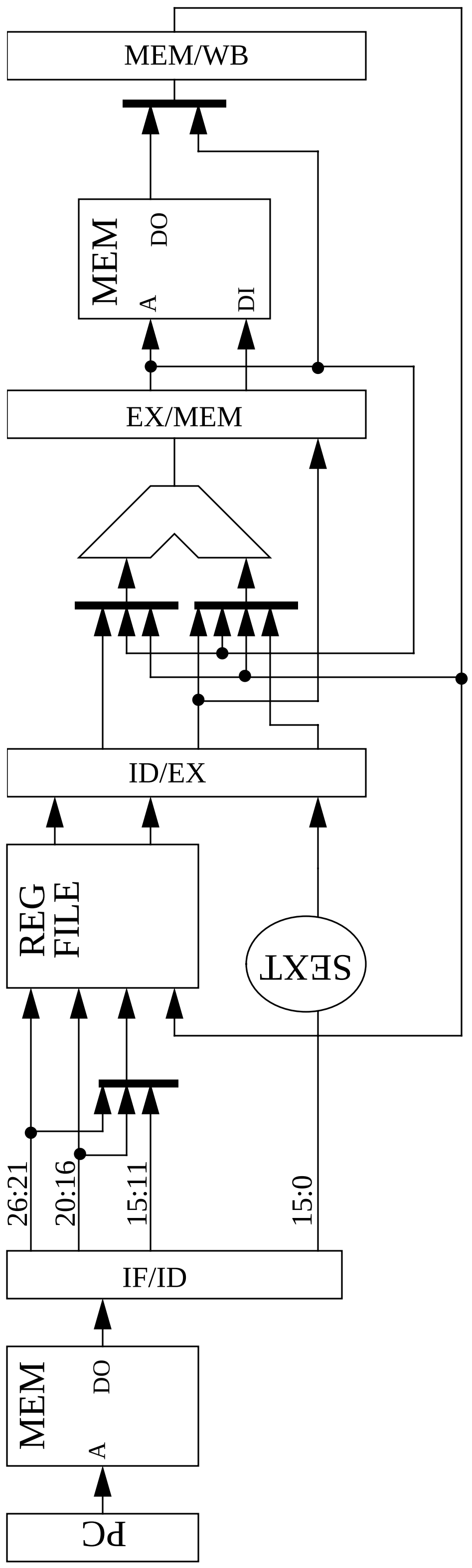
c) In the old pipeline, $CPI_{load} = 1.6$, indicating that 60% of the loads have a bubble that can't be filled. However, for the new machine, $CPI_{load} = 1$, since loads can no longer cause pipeline bubbles. The average CPI is just the weighted sum of the individual instruction CPIs, thus

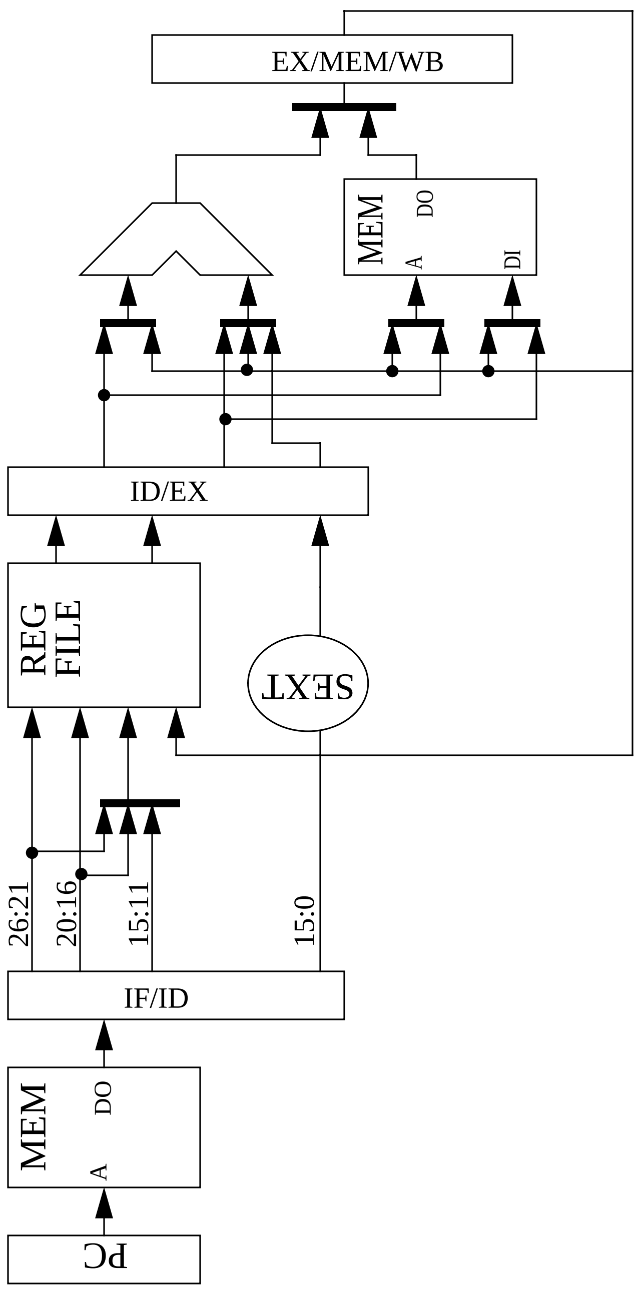
$$CPI_{old} = 0.4 + 0.3*1.6 + 0.15 + 0.15*2 = 1.33$$

$$CPI_{new} = 0.45 + 0.274 + 0.138 + 0.138*2 = 1.14$$

*d) Recall the performance equation: T = IC*CPI*CCT. We assume that CCT doesn't change (a safe assumption. Speedup is $T_{old}/T_{new} = (IC_{old}*CPI_{old})/(IC_{new}*CPI_{new}) = (1/1.09)(1.33/1.14) = 1.07$.*

e. Below is a simplified pipelined datapath that implements the base MIPS instruction set. Draw the modifications required to implement the new four stage pipeline from above. Explain any modifications, as necessary to make your response clear. *(5 pts)*





8. While you have worked quite a bit with a simple 5-stage pipelined processor, this question asks you to analyze a slightly more complicated pipeline. The pipeline diagram is shown below and contains the normal read (R), execute (X), and write-back (W) stages. Note however, that instruction fetch takes two cycles (F1 and F2) and memory access takes two cycles (M1 and M2). Also note that we have pipelined the multiply into 3 stages (X1, X2, and X3).

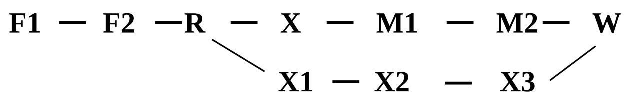
   a. For the MIPS assembly code shown below, how many cycles will it take to execute the loop in steady state. You may assume a predict branch not-taken policy (no delay slots), but state any other assumptions. Explain and show your work for full credit. *(5 pts)*

   b. How much better could you do with a dynamic branch predictor? State your assumptions and explain. *(5 pts)*

   c. What is the CPI of this code sequence (assuming that the pipeline starts out full and you use a dynamic branch predictor)? *(5 pts)*

   d. If the deeper pipelining allows us to double the clock frequency, what is the speedup relative to the original MIPS 5 stage pipeline that also uses a dynamic branch predictor? *(5 pts)*

   e. How much better could you do for the pipeline below by optimizing the code to avoid the hazards? *(5 pts)*



```
.text

_loop:
(1)  lw $t0, 0($t2)
(2)  lw $t1, 0($t3)
(3)  mul $t1, $t0, $t1
(4)  sw $t1, 0($t4)

(5)  lw $t0, 4($t2)
(6)  lw $t1, 4($t3)
(7)  add $t1, $t0, $t1
(8)  sw $t1, 4($t4)

(9)  add $t2, $t2, 4
(10) add $t3, $t3, 4
(11) add $t4, $t4, 4
(12) slt $t5, $t4, $t7
(13) bne $t5, $zero, _loop

_end:
```

a) To solve this problem, we merely need to analyze one iteration to determine the number of pipeline bubbles due to hazards. In this case, we'll have 2 after (2), 1 after (3), 2 after (6), and 2 after (13) because the branch is usually taken. Adding these 8 cycles to the 13 instructions give a total of 20 cycles in steady state.

b) A dynamic branch predictor could easily learn the branch to be taken and if the number of iterations is high enough, prediction accuracy would be near perfect. This would reduce the number of pipeline bubbles to 6 and the total cycle count to 18, a 10% improvement.

c) The CPI is merely 18/13 = 1.38.

*d) Again, T = IC*CPI*CCT. IC doesn't change and* $CCT_{new} = 0.5*CCT_{old}$*. Now we need to compute* CPI_{old}*. Using the same analysis as above, the number of pipeline bubbles (assuming dynamic branch prediction) is just two (one each for instructions 2 and 6). Thus* $CPI_{old} = 15/13 = 1.15$*. Finally*

$$Speedup = (CPI_{old}*CCT_{old})/(CPI_{new}*CCT_{new})$$
$$= (1.15/1.46)*(1/0.5) = 1.67.$$

We can also say that the new machine is 40% faster.

Old Code

```
.text

_loop:
(1)  lw $t0, 0($t2)
(2)  lw $t1, 0($t3)
(3)  mul $t1, $t0, $t1
(4)  sw $t1, 0($t4)

(5)  lw $t0, 4($t2)
(6)  lw $t1, 4($t3)
(7)  add $t1, $t0, $t1
(8)  sw $t1, 4($t4)

(9)  add $t2, $t2, 4
(10) add $t3, $t3, 4
(11) add $t4, $t4, 4
(12) slt $t5, $t4, $t7
(13) bne $t5, $zero, _loop

_end:
```

New Code

```
.text

_loop:
(5)  lw $t6, 4($t2)
(6)  lw $t8, 4($t3)
(1)  lw $t0, 0($t2)
(7)  add $t8, $t6, $t8

(2)  lw $t1, 0($t3)
(8)  sw $t8, 4($t4)
(3)  mul $t1, $t0, $t1
(9)  add $t2, $t2, 4
(10) add $t3, $t3, 4
(4)  sw $t1, 0($t4)

(11) add $t4, $t4, 4
(12) slt $t5, $t4, $t7
(13) bne $t5, $zero, _loop

_end:
```

e) With a little surgery, the instructions can be arranged to cover all of the load and multiply delay slots, as shown in the new code on the right. Note that it did require some additional registers ($t6 and $t8) because the new organization needs to keep more live values in registers. Such is the trade-off between register pressure and concurrency (more on this later). With this new code, the CPI improves to 1 and the overall speedup improves from 1.58 to 2.

Note that there are other ways of optimizing this code to achieve the same result.