15740-18740 Computer Architecture, Fall 2010
Midterm Exam I

Instructor: Onur Mutlu
Teaching Assistants: Evangelos Vlachos, Lavanya Subramanian, Vivek Seshadri
Date: October 11, 2010

Name:

Problem 1 (50 points)	:	
Problem 2 (15 points)	:	
Problem 3 (12 points)	:	
Problem 4 (25 points)	:	
Problem 5 (15 points)	:	
Problem 6 (20 points)	:	
Problem 7 (25 points)	:	
Bonus Problem 8 (10 points)	:	
Legibility and Name (3 points)	:	
Total (165 + 10 points)	:	

Instructions:

1. This is a closed book exam. You are allowed to have one letter-sized cheat sheet.
2. No electronic devices may be used.
3. This exam lasts 1 hour 50 minutes.
4. Clearly indicate your final answer.
5. Please show your work when needed.
6. Please write your name on every sheet.
7. Please make sure that your answers to all questions (and all supporting work that is required) are contained in the space provided.

Problem 1 (50 points)

1) Pipelining (5 points)

Give two reasons why IPC could decrease when we increase the pipeline depth of a processor, if this is actually possible. Explain very briefly.

Branch misprediction penalty increases

Pipeling bubbles due to data dependencies increase

Give two reasons why IPC could increase when we increase the pipeline depth of a processor, if this is actually possible. Explain very briefly.

Never. Increasing pipeline depth cannot increase IPC

2) Interrupts & Exceptions (6 points)

Assume that a floating point instruction generates a divide by zero exception during runahead mode in a runahead execution processor. When is this exception handled? Explain briefly.

Never. Runahead mode is purely speculative.

Assume that a load instruction addresses a memory location that is not present in physical memory in runahead mode. When is this exception handled? Explain briefly.

Never. Runahead mode is purely speculative.

Assume that the keyboard generates an interrupt request to the processor during runahead mode. When is this interrupt handled? Explain briefly.

When convenient. The keyboard interrupt is not very time critical.

3) Instruction & Data Caches (4 points)

An engineer designs a deeply pipelined, 30-stage machine. The goal is to achieve very high clock frequency, while having reasonable IPC. She is considering two design options for the first-level caches:

1. A unified instruction+data cache that has a total of 64KB data storage.
2. A separate 32KB instruction cache and 32KB data cache.

Describe which option you would recommend and why, in less than 30 words.

The second option. Instruction and data caches are accessed during different stages of the pipeline. Separate I and D caches eliminate resource contention and reduce the delay in accessing the caches.

4) TLB Management (6 points)

Some ISAs (e.g., MIPS) specify a TLB miss to be handled by the operating system instead of hardware. This is called a software managed TLB, as opposed to a hardware-managed one, in which the hardware performs the TLB miss handling.

a) Describe one advantage of a software managed TLB over a hardware managed one (less than 20 words please).

More flexible page table organization and TLB replacement

b) Describe one advantage of a hardware managed TLB over a software managed one (less than 20 words please).

It is faster

c) Suppose a sleep-deprived engineer proposes handling TLB hits in software, in the wake of a eureka moment that comes after drinking many cups of coffee. Is this a good idea or a bad idea? Explain in no more than 20 words.

Bad idea. This would cause each TLB hit to take many cycles.

5) Branch Handling (4 points)

a) In what sense is the handling of a branch misprediction similar to the handling of an exception?

Both branch misprediction and an exception need the processor state to be restored to a precise point so that they can be handled.

b) In what sense is the handling of a branch misprediction different from the handling of an exception?

1. Branch mispredictions occur more frequently than exceptions. Hence, they have to be handled fast.
2. Branch mispredictions redirect control flow to the dynamic target of the branch rather than a software handler.
3. Branch mispredictions are not visible to software.

6) Runahead, In-order, Out-of-order Execution (4 points)

The idea of runahead execution, as discussed in class, can be employed in conjunction with in-order instruction scheduling or out-of-order instruction scheduling. We have discussed results from multiple different systems showing that the relative performance benefit of runahead execution is higher when it is implemented on top of a machine that employs in-order scheduling than when it is implemented on top if a machine that employs out-of-order instruction scheduling. Explain why this is so. (20 words or less)

In-order execution has less tolerance to memory latency than out-of-order execution.

7) Asymmetric CMPs (5 points)

A multi-core processor consists of 16 simple cores on a single chip. A program consists of code that is 80% perfectly parallelizable and 20% serial.
a) What is the maximum speed-up obtained by executing this program on the 16-core chip versus running it on a chip that only has a single simple core?

Speed-up $= \frac{1}{0.2+\frac{0.8}{16}} = \frac{1}{0.25} = 4$

b) Suppose instead we execute this program on a chip where 4 of the 16 simple cores have been replaced by one heavyweight core that processes instructions 2 times faster than a simple core. What is the speed-up of running on this chip versus running on a chip that has 16 simple cores? Assume that when the chip is processing the parallelizable code, the heavyweight core is idle.

Speed-up over single simple core $= \frac{1}{\frac{0.2}{2}+\frac{0.8}{12}} = \frac{3}{0.5} = 6$
Therefore, speed-up over 16 simple core chip $= \frac{6}{4} = 1.5$

8) ISA Tradeoffs (8 points)

Embedded systems usually have relatively small physical memories due to size and power constraints. Assume that you are designing such an embedded system from scratch with a very small physical memory. What are some of the design choices you will make with respect to the following parameters? For each parameter, specify which of the choices you would pick with a brief explanation. If there are trade-offs, clearly mention them in your answer.

a) Complex instructions or simple instructions

Complex instructions: We would like all instructions to perform a large amount of work as this will reduce code size in physical memory.

b) Many architectural registers or few architectural registers

It depends. Many architectural registers reduce the number of load or store instructions required to save or restore registers when there is register pressure, but they increase instruction sizes. Few architectural registers lead to smaller instruction sizes, but cause register pressure which leads to additional load or store instructions.

c) 64-bit registers or 32-bit registers

It depends. 64-bit registers can reduce the number of instructions that manipulate data (e.g., only one instruction is used to XOR two 32-bit data elements), thereby reducing physical memory size. 32-bit registers reduce the space occupied by each thread context in physical memory.

d) Virtual memory or no virtual memory

No virtual memory: Because virtual memory requires additonal page tables which need to be stored in physical memory.

e) Small pages or large pages (assuming you have virtual memory)

It depends. Large pages imply more internal fragmentation (i.e. need for larger physical memory). Small pages imply larger page tables (i.e. need for larger physical memory).

9) L1 Caches and Virtual Memory (8 points)

We wish to design a 64 KB L1 cache for an ISA having a 4 KB page size. We want to keep the TLB out of the critical path of cache access, but also ensure that a given physical address can reside in only one unique location in the cache without any software support. What is the associativity of our design that satisfies the above constraints? Show your work.

Without any software support, we need the index bits not to change during translation. Therefore,
Minimum associativity $= \frac{\textbf{Cache size}}{\textbf{Page Size}} = \frac{64}{4} = 16$

Suppose we relax the requirement of not having software support above, but we would still like to satisfy the other above constraints. In addition, we would like to build the cache as direct mapped. What does the system software need to ensure such that a given physical address resides in only one unique location in the cache? Be specific. Show your work.

In this case, the software should ensure that the index bits do not change during translation. So the operating system should ensure that the last four bits of the virtual page number are the same ad the last four bits of the corresponding physical frame number.

Problem 2 (Interlocking) (15 points)

a) In lecture, we discussed the motivation behind MIPS, which stands for Microprocessor without Interlocked Pipe Stages. The motivation for MIPS was to keep the hardware design as simple as possible by eliminating the logic that checks for dependencies between instructions. What was the key idea to eliminate hardware interlocks used in early MIPS designs? Explain.

The compiler ensures that no two dependent instructions are in the pipeline at the same time.

b) Fine-grained multithreading (FGMT), also discussed in lecture, and employed in Denelcor HEP and Sun Niagara is another alternative method for eliminating the need for hardware-based interlocking. Explain how this works briefly. Hint: Describe the invariant fine-grained multithreading enforces to eliminate hardware interlocks.

The hardware ensures that no two instructions from the same thread exists in the pipeline concurrently (i.e. fetch from a thread once every N cycles)

c) Describe two advantages of the MIPS approach over FGMT.

1. Better single thread performance.
2. The hardware design can be simple, no need to fetch from or have multiple registers for multiple hardware contexts

d) Describe two advantages of FGMT over the MIPS approach.

1. Simpler compiler
2. Assuming multiple independent threads are available, FGMT potentially better utilizes the pipeline
3. Better memory latency tolerance.

e) An alternative way to eliminate the need for interlocking is to predict the unavailable source values of an instruction. What additional logic does this approach require that FGMT and the MIPS approach do not require?

Logic to predict data values. Logic to recoved from value mispredictions

Problem 3 (Mispredicted Path and Memory) (12 points)

An engineer designs a piece of logic that, upon discovering that a branch has been mispredicted, cancels all memory requests that were initiated by load instructions along the mispredicted path. We evaluate this new feature on application A, and notice a performance improvement. On application B, we notice a performance degradation. There is nothing wrong with the measurement methodology or hardware implementation. Explain how both results are possible. Show code examples, if you think this would aid the explanation.

In application A, cache blocks fetched by memory requests generated on the mispredicted path are later not needed by the correct path execution. These requests can cause cache pollution by evicting useful cache blocks or can delay correct-path requests by occupying memory bandwidth and buffers. As such, eliminating them improves performance.
In application B, cache blocks fetched by memory requests generated on the mispredicted path are later needed by the correct path execution. In other words, execution on the mispredicted path would prefetch data and instructions that will later be used on the correct path. As such, eliminating mispredicted memory requests reduces performance because it would eliminate the benefits of such prefetching.

Problem 4 (Runahead) (25 points)

We have covered the notion of runahead execution in class, as an alternative to scaling up the instruction window size of an out-of-order execution processor. Remember that a runahead processor speculatively executes the program under an L2 cache miss. Also remember that the instruction window determines how many instructions an out of order execution engine contains that are decoded but not yet committed to architectural state.

a) What benefit(s) of a large instruction window can runahead execution provide? Be specific.

Memory Level Parallelism (MLP). The ability to parallelize multiple L2 cache misses.

b) What benefit(s) of a large instruction window can runahead execution not provide? Be specific.

The ability to execute instructions other than cache misses in parallel.

c) Remember that runahead mode ends when the L2 cache miss that caused entry into runahead mode is fully serviced by main memory. How does the processor exit runahead mode? Describe the actions taken by the processor.

1. **Flush the pipeline**
2. **Restore the architectural state to checkpoint**
3. **Restore Program Counter to the runahead causing load**
4. **Start execution from the runahead causing load**

d) Where does the processor start execution after runahead mode?

From the runahead causing load

e) Two computer architects debate whether it is a good idea to exit runahead mode when the L2 cache miss that caused entry into runahead mode (i.e., the runahead-causing miss) is fully serviced. Each comes up with a different suggestion. Vivek suggests that the processor stay 20 more cycles after the runahead-causing miss is fully serviced. Explain why this could be a good idea (i.e., when could it provide benefits).

The processor can initiate a cache miss in those 20 cycles.

f) Explain why Vivek's suggestion could be a bad idea (i.e., when could it do harm or not provide benefits).

The processor cannot find any long latency loads to initiate in those 20 cycles. So, it wastes time instead of doing useful work.

g) Lavanya, on the other hand, develops a cheap mechanism that enables the processor to detect that the runahead-causing cache miss will be fully serviced in 20 cycles. Using this mechanism, Lavanya suggests that the processor exit runahead mode 20 cycles before the runahead-causing miss is serviced. Explain why this could be a good idea.

The processor can overlap runahead exit penalty in these 20 cycles while the miss comes back.

h) Explain why Lavanya's suggestion could be a bad idea.

1. The processor could have discovered more misses in these 20 cycles
2. Extra hardware and complexity

i) A third architect, Evangelos, listens to both and develops a mechanism that aims to achieve the best of both solutions. Briefly describe what such a mechanism could be. Focus on key ideas (not implementation), but be specific (i.e., the mechanism needs to be implementable; it should not assume the existence of information whose collection is not possible).

Idea: Predict if there will be cache misses in the 40 cycle window.
One possibility: Check how many misses are encountered in the runahead mode so far. If this is less than a threshold, exit runahead mode 20 cycle early, if it is greater than a threshold, exit runahead mode 20 cycles late. This mechanism predics that the more misses we encounter, the more misses we are likely to encounter.

j) Explain why the mechanism you propose could be a good idea.

This mechanism could adapt to application behavior

k) Explain why the mechanism you propose could be a bad idea.

1. A misprediction leads to performance degradation
2. Additional complexity of the predictor

Problem 5 (Caches) (15 points)

A byte-addressable system with 16-bit addresses ships with a two-way set associative, write back cache with perfect LRU replacement. The tag store requires a total of 4352 bits of storage. What is the block size of the cache? Show all your work.
(Hint: $4352 = 2^{12} + 2^8$)

Assume t tag bits, n index bits and b block bits.

$$\begin{aligned}
t + n + b &= 16 \\
\textbf{LRU} &= \textbf{1 bit per set} \\
\textbf{Valid Bit} &= \textbf{1 bit per block} \\
\textbf{Dirty Bit} &= \textbf{1 bit per block} \\
\textbf{Tag bits} &= t\ \textbf{bits per block} \\
\textbf{Number of sets} &= 2^n \\
\textbf{Number of blocks} &= 2 \times 2^n \quad \textbf{(2-way associative)} \\
\textbf{Tag store size} &= 2^n + 2 \times 2^n \times (2 + t) \\
\Rightarrow 2^n \times (2t + 5) &= 2^{12} + 2^8 \\
\Rightarrow 2^n \times (2t + 5) &= 2^8 \times 17 \\
\Rightarrow n = 8 \quad &\textit{and} \quad t = 6 \\
\Rightarrow b = 2 &
\end{aligned}$$

Therefore, the block size is 4 bytes.

Problem 6 (Out-of-order Core Design) (20 points)

In lectures, we described the design of out-of-order execution machines in which register data values are consolidated in a single structure, called the physical register file (PRF). This approach is used in several modern out-of-order execution processors, including the Pentium 4, Alpha 21264, and MIPS R10000. Let's call the approach the "PRF design." We have contrasted this approach to the approach of storing register values in reservation stations, reorder buffer, future register file, and architectural register file. The latter approach is used in the Pentium Pro design. Let's call this approach the "PPro Design."

a) At what stage during its execution does an instruction write its result to the physical register file in the PRF design?

After it completes execution

At the same stage, where does the instruction write its result in the PPro design?

To the reorder buffer. (Many of you wrote future file, reservation station. Note that the PPro design broadcasts the tag/value to the future file and the reservation station. This does not necessarily mean that the result gets written to them. Only if there is a tag match, the result is written.

b) At what stage during its execution does an instruction write its result to the architectural register file in the PPro design?

Commit Stage (when the instruction has completed execution and is the oldest in the reorder buffer

At the same stage, where does the instruction write its result in the PRF design?

Nowhere. The result is already written to the the physical register file.

c) How does the architectural register state get updated in each design when an instruction commits? Explain briefly.

PRF:

Update the architectural (retirement) register map for the destination architectural register to point to the physical register allocated for the instruction.

PPro:

Copy the result from the re-order buffer to the architectural register file.

d) When does a physical register get deallocated (i.e., freed) in the PRF design? Explain.

1. When an architectural register pointing to the physical register gets overwritten with another physical register ID
2. When the pipeline gets flushed and the physical register was allocated for some re-order buffer entry

e) What are the advantages of the PRF design over the PPro design?

1. Avoid redundant storage of values. PRF contains all the values.
2. Avoid broadcasting values through the common data bus.

f) What are the disadvantages of the PRF design over the PPro design?

1. Register access requires a separate stage (indirection)
2. Physical register file should be much bigger than architectural register file (increases latency)
3. Register allocation/deallocation is slightly more complex.
4. Cannot access the architectural register file directly.

Problem 7 (Out of Order Execution) (25 points)

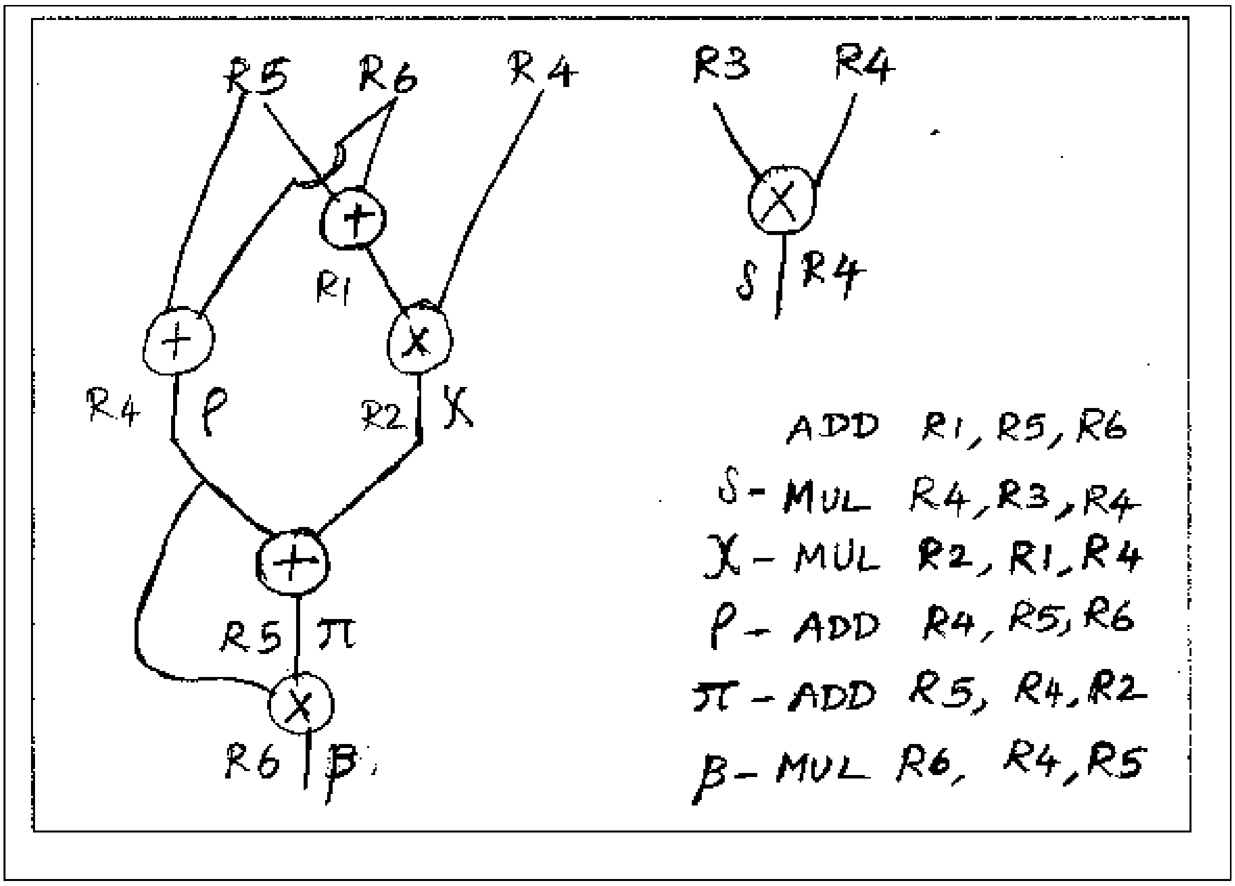
The diagram below shows a snapshot at a particular point in time of various parts of the microarchitecture for an implementation supporting out-of-order execution in the spirit of Tomasulo's algorithm.

Note that both the adder and multiplier are pipelined, the adder has three stages and the multiplier has four stages. At the particular point in time when the snapshot was taken, the adder had only one instruction in the pipeline (i.e., in the final stage). The multiplier had two instructions in the pipeline, one in the final stage, one in stage 3. Note that the multiplier and adder have separate output data buses, which allow both an add result and a multiply result to update in the same cycle.

The adder has two reservation stations, the multiplier has four reservation stations. An instruction continues to occupy a reservation station slot until it completes execution and its result is written to its destination register.

The processor has been supplied with a six instruction program segment. Instructions are fetched, decoded and executed as discussed in class. Assume full data forwarding, whenever possible.

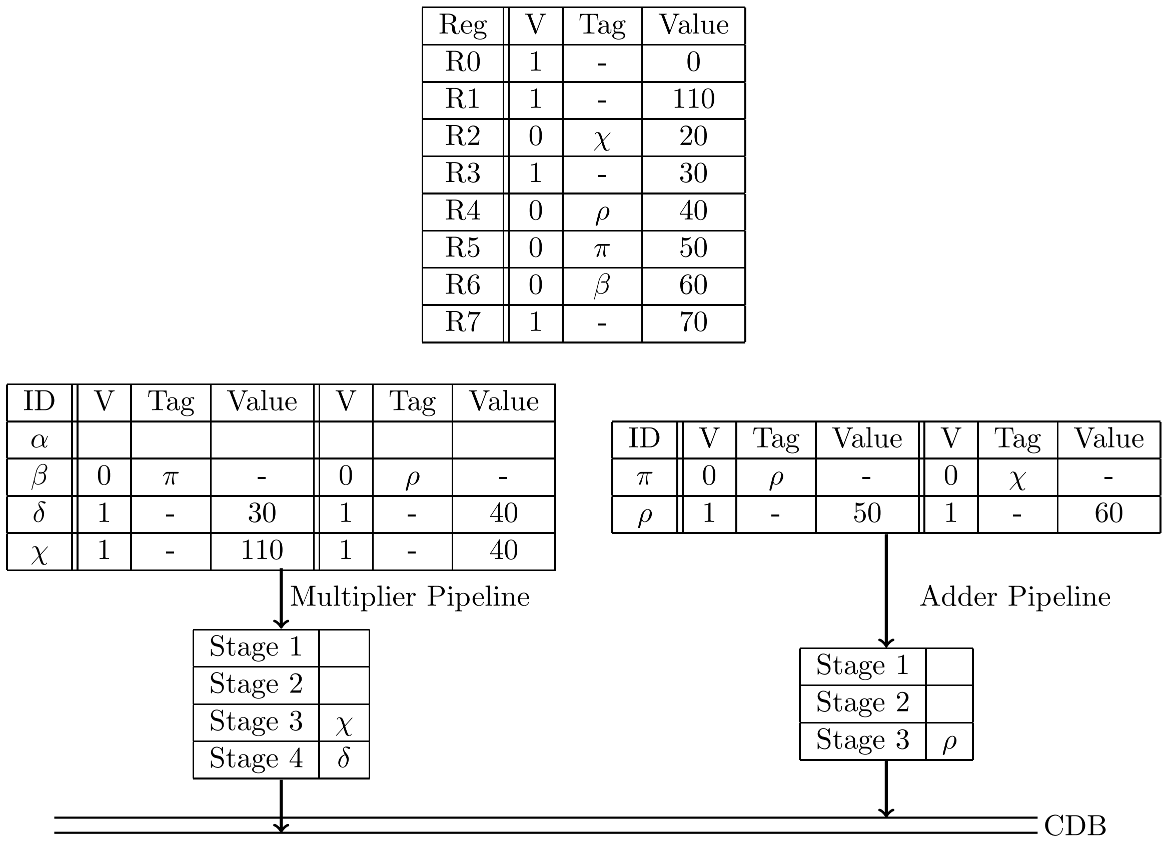
Your job: First, identify the instructions and draw the data flow graph for the six instructions. Use + for `ADD` and × for `MULTIPLY`.



```
    ADD  R1, R5, R6
δ - MUL  R4, R3, R4
χ - MUL  R2, R1, R4
ρ - ADD  R4, R5, R6
π - ADD  R5, R4, R2
β - MUL  R6, R4, R5
```

Second, complete the table identifying the six instructions, their opcodes (`ADD`, `MUL`), and their register operands ($\texttt{R}_{\texttt{dest}}$, $\texttt{R}_{\texttt{src1}}$, $\texttt{R}_{\texttt{src2}}$). The first instruction is already filled for you: `ADD R1, R5, R6`. (`R1` is the destination register and `R5, R6` are the source registers.) Also, show which stage of processing (F, D, E, R, W) each instruction is in during each cycle. Assume a 5-stage pipeline: `Fetch(F), Decode(D), Execute(E), Reorder Buffer(R), Writeback(W)`, as we covered in lecture. If an instruction is stalled, indicate it with a '–'.

Finally, identify the cycle after which the snapshot of the microarchitecture was taken. Shade the corresponding cycle in the last row of the table.



Reg	V	Tag	Value
R0	1	-	0
R1	1	-	110
R2	0	χ	20
R3	1	-	30
R4	0	ρ	40
R5	0	π	50
R6	0	β	60
R7	1	-	70

ID	V	Tag	Value	V	Tag	Value
α						
β	0	π	-	0	ρ	-
δ	1	-	30	1	-	40
χ	1	-	110	1	-	40

ID	V	Tag	Value	V	Tag	Value
π	0	ρ	-	0	χ	-
ρ	1	-	50	1	-	60

Figure 1: Snapshot of the Register file, Reservation stations and the Pipelines

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
`ADD R1,R5,R6`	F	D	E	E	E	R	W															
`MUL R2,R1,R4`		F	D	-	-	E	E	E	E	R	W											
`MUL R4,R3,R4`			F	D	E	E	E	E	R	-	-	W										
`ADD R4,R5,R6`				F	D	E	E	E	R	-	-	-	W									
`ADD R5,R4,R2`					F	D	-	-	-	E	E	E	R	W								
`MUL R6,R4,R5`						F	D	-	-	-	-	-	E	E	E	E	R	W				
Snapshot Cycle								■														

Table 1: **THIS IS THE TABLE FOR YOU TO FILL OUT**

Bonus Problem 8 (Good Enough Computing) (10 points)

Ben Zorn spoke about the concept of "good enough" computing and mentioned the idea of approximate loop computing. Consider the following program:

```
int a[1000000];
int avg = 0;

for (int k = 0; k < 1000000; k ++) {
   avg += a[k];
}

avg /= 1000000;
```

Assume that the array 'a' is not in cache and needs to be accessed from main memory. If we are interested in just computing the average (avg) *quickly*, how could you modify the above code in the context of "Good enough" computing as suggested by Ben Zorn? How will you provide an estimate of how good the output is?

1. Skip iterations from the loop. Keep track of how many iterations you have skipped to compute the average and goodness of the result.
2. Random sampling. Sample the numbers based on some distribution (e.g. uniform distribution). Keep track of the number of samples to recompute the average and goodness.

If we were to annotate the data in the program to distinguish critical data (remember Ben's lecture) from non-critical data, which of the variables will you mark as critical?

1. `a`: We do not want the base pointer to be corrupted.
2. `k`: The iteration variable is critical to staying within bounds.
3. `avg`: Having the higher order bits of average corrupted can lead to very inaccurate results.

When does "good enough" computing make sense? When does it not? Explain briefly.

Good enough computing makes sense when you can gain significant performance or energy savings for slight loss in accuracy. For this, we should be able to estimate the goodness of the inaccurate computation when compared to an accurate computation. Good enough computing does not make sense for mission critical applications where accuracy is really important.