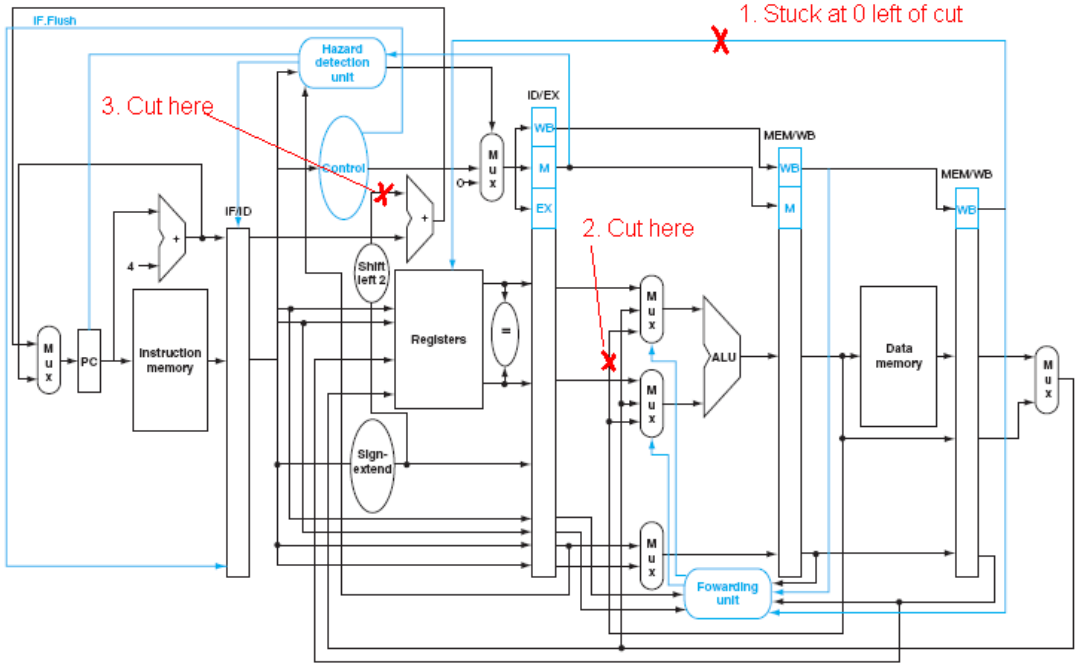
Practice Final

[6 points] For the MIPS datapath shown below, several lines are marked with "X". For each one:

- Describe in words the negative consequence of cutting this line relative to the working, unmodified processor.
- Provide a snippet of code that will fail
- Provide a snippet of code that will still work



3. [5 points] Structural, data and control hazards typically require a processor pipeline to stall. Listed below are a series of optimization techniques implemented in a compiler or a processor pipeline designed to reduce or eliminate stalls due to these hazards. For each of the following optimization techniques, state which pipeline hazards it addresses and how it addresses it. Some optimization techniques may address more than one hazard, so be sure to include explanations for all addressed hazards.

(a) Branch Prediction

(b) Instruction Scheduling

(c) delay slots

(d) increasing availability of functional units (ALUs, adders etc)

(e) caches

5. [16 points] The classic 5-stage pipeline seen in Section 4.5 is IF, ID, EX, MEM, WB. This pipeline is designed specifically to execute the MIPS instruction set. MIPS is a load store architecture that performs one memory operation per instruction, hence a single MEM stage in the pipeline suffices. Also, its most common addressing mode is register displacement addressing. The EX stage is placed before the MEM stage to allow it to be used for address calculation. In this question we will consider a variation in the MIPS instruction set and the interactions of this variation with the pipeline structure.

The particular variation we are considering involves swapping the MEM and EX stages, creating a pipeline that looks like this: IF, ID, MEM, EX, WB. This change has two effects on the instruction set. First, it prevents us from using register displacement addressing (there is no longer an EX in front of MEM to accomplish this). However, in return we can use instructions with one memory input operand, i.e., register-memory instructions. For instance: `multf_m f0,f2,(r2)` multiplies the contents of register `f2` and the value at memory location pointed to by `r2`, putting the result in `f0`.

(a) Dropping the register displacement addressing mode is potentially a big loss, since it is the mode most frequently used in MIPS. Why is it so frequent? Give two popular software constructs whose implementation uses register displacement addressing (i.e., uses displacement addressing with non-zero displacements).

(b) What is the difference between a dependence and a hazard?

(c) In this question we will work with the SAXPY loop.

```
do I = 0,N
Z[I] = A*X[I] + Y[I]
```

Here is the new assembly code.

```
0: slli r2,r1,#3 // I is in r1
1: addi r3,r2,#X
2: multf_m f2,f0,(r3) // A is in f0
3: addi r4,r2,#Y
4: addf_m f4,f2,(r4)
5: addi r4,r2,#Z
6: sf f4,(r4)
7: addi r1,r1,#1
8: slei r6,r1,r5 // N is in r5
9: bnez r6,#0
```

Using the instruction numbers, label the data and control dependences.

(d) Fill in the pipeline diagram for code for the new SAXPY loop. Label the stalls as d* for data-hazard stalls and s* for structural stalls. What is the latency of a single iteration? (The number of cycles between the completion of two successive #0 instructions). For this question, assume that FP addition takes 2 cycles, FP multiplication takes 3 cycles and that all other operations take a single cycle. The functional units are not pipelined. The FP adder, FP multiplier and integer ALU are all separate functional units, such that there are no structural hazards between them. The register file is written by

the WB stage in the first half of a clock cycle and is read by the ID stage in the second half of a clock cycle. In addition, the processor has full forwarding. The processor stalls on branches until the outcome is available which is at the end of the EX stage. The processor has no provisions for maintaining "precise state".

instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0: slli r2,r1,#3	F	D	M	X	W																			
1: addi r3,r2,#X		F	D	M	X	W																		
2: mulf_m f2,f0,(r3)			F	D		M	X	X	X	W														
3: addi r4,r2,#Y				F		D	M	X	W															
4: addf_m f4,f2,(r4)						F	D		M	X	X	W												
5: addi r4,r2,#Z							F		D	M	X		W											
6: sf f4,(r4)									F	D		M	X	W										
7: addi r1,r1,#1										F		D	M	X	W									
8: slei r6,r1,r5												F	D	M	X	W								
9: bnez r6, #0													F	D	M	X	W							
0: slli r2,r1,#3																	F	D	M	X	W			

(e) In the pipeline for MIPS mentioned in the text, what is the reason for forcing non-memory operations to go through the MEM stage rather than proceeding directly to the WB stage?

(f) Aside from the direct loss of register displacement addressing and the subsequent instructions required to explicitly compute addresses, what are two other disadvantages of this sort of pipeline?

(g) Reduce the stalls by pipeline scheduling a single loop iteration. Show the resulting code and fill in the pipeline diagram. You do not need to show the optimal schedule for a correct response.

instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0: slli r2,r1,#3	F	D	M	X	W																			
1: addi r3,r2,#X		F	D	M	X	W																		
2: mulf_m f2,f0,(r3)			F	D		M	X	X	X	W														
3: addi r4,r2,#Y				F		D	M	X	W															
4: addf_m f4,f2,(r4)						F	D		M	X	X	W												
5: addi r4,r2,#Z							F		D	M	X		W											
6: sf f4,(r4)									F	D		M	X	W										
7: addi r1,r1,#1										F		D	M	X	W									
8: slei r6,r1,r5												F	D	M	X	W								
9: bnez r6, #0													F	D	M	X	W							
0: slli r2,r1,#3																	F	D	M	X	W			

6. [8 points] A two-part question.

(Part A) Dependence detection

This question covers your understanding of dependences between instructions. Using the code below, list all of the dependence types (RAW, WAR, WAW). List the dependences in the respective table (example INST-X to INST-Y) by writing in the instruction numbers involved with the dependence.

```
I0: A = B + C;
I1: C = A - B;
I2: D = A + C;
I3: A = B * C * D;
I4: C = F / D;
I5: F = A ^ G;
I6: G = F + D;
```

RAW Dependence		WAR Dependence		WAW Dependence	
From Instr	To Instr	From Instr	To Instr	From Instr	To Instr

(Part B) Dependence analysis

Given four instructions, how many unique comparisons (between register sources and destinations) are necessary to find all of the RAW, WAR, and WAW dependences. Answer for the case of four instructions, and then derive a general equation for N instructions. Assume that all instructions have one register destination and two register sources.

3. [6 points] A two part question

(a) Why is miss rate not a good metric for evaluating cache performance? What is the appropriate metric? Give its definition. What is the reason for using a combination of first and second- level caches rather than using the same chip area for a larger first-level cache?

(b) The original motivation for using virtual memory was "compatibility". What does that mean in this context? What are two other motivations for using virtual memory?

9. [3 points] Pipelining is used because it improves instruction throughput. Increasing the level of pipelining cuts the amount of work performed at each pipeline stage, allowing more instructions to exist in the processor at the same time and individual instructions to complete at a more rapid rate. However, throughput will not improve as pipelining is increased indefinitely. Give two reasons for this.

4. [6 points] A four part question

(Part A)

What are the two characteristics of program memory accesses that caches exploit?

(Part B)

What are three types of cache misses?

Cold misses, conflict misses and compulsory misses

(Part C)

Design a 128KB direct-mapped data cache that uses a 32-bit address and 16 bytes per block. Calculate the following:

(a) How many bits are used for the byte offset?

(b) How many bits are used for the set (index) field?

(c) How many bits are used for the tag?

8. [12 points] A three-part question. This question covers cache and pipeline performance analysis.

(Part A)

Write the formula for the average memory access time assuming one level of cache memory:

(Part B)

For a data cache with a 92% hit rate and a 2-cycle hit latency, calculate the average memory access latency. Assume that latency to memory and the cache miss penalty together is 124 cycles. Note: The cache must be accessed after memory returns the data.

(Part C)

Calculate the performance of a processor taking into account stalls due to data cache and instruction cache misses. The data cache (for loads and stores) is the same as described in Part B and 30% of instructions are loads and stores. The instruction cache has a hit rate of 90% with a miss penalty of 50 cycles. Assume the base CPI using a perfect memory system is 1.0. Calculate the CPI of the pipeline, assuming everything else is working perfectly. Assume the load never stalls a dependent instruction and assume the processor must wait for stores to finish when they miss the cache. Finally, assume that instruction cache misses and data cache misses never occur at the same time. Show your work.

- Calculate the additional CPI due to the icache stalls.
- Calculate the additional CPI due to the dcache stalls.
- Calculate the overall CPI for the machine.

9. [12 points] A three-part question.

(Part A)

A processor has a 32 byte memory and an 8 byte direct-mapped cache. Table 0 shows the current state of the cache. Write hit or miss under the each address in the memory reference sequence below. Show the new state of the cache for each miss in a new table, label the table with the address, and circle the change:

Addr	10011	00001	00110	01010	01110	11001	00001	11100	10100
H/M									

0. Initial state

Index	V	Tag	Data
000	N		
001	Y	00	Mem(00001)
010	N		
011	Y	11	Mem(11011)
100	Y	10	Mem(10100)
101	Y	01	Mem(01101)
110	Y	00	Mem(00110)
111	N		

1.

Index	V	Tag	Data
000			
001			
010			
011			
100			
101			
110			
111			

2.

Index	V	Tag	Data
000			
001			
010			
011			
100			
101			
110			
111			

3.

Index	V	Tag	Data
000			
001			
010			
011			
100			
101			
110			
111			

4.

Index	V	Tag	Data
000			
001			
010			
011			
100			
101			
110			
111			

5.

Index	V	Tag	Data
000			
001			
010			
011			
100			
101			
110			
111			

6.

Index	V	Tag	Data
000			
001			
010			
011			
100			
101			
110			
111			

7.

Index	V	Tag	Data
000			
001			
010			
011			
100			
101			
110			
111			

(Part B)

Do the same thing as in Part A, except for a 4-way set associative cache. Assume 00110 and 11011 were the last two addresses to be accessed. Use the Least Recently Used replacement policy.

Addr	10011	00001	00110	01010	01110	11001	00001	11100	10100
H/M									

0.

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0			0011	Mem(00110)			1010	Mem(10100)
1	0000	Mem(00001)	1101	Mem(11011)	0110	Mem(01101)		

1.

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

2.

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

3.

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

4.

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

5.

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

6.

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

7.

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

12. [10 points] Suppose we have a dual core chip multiprocessor with two level cache hierarchy: Both the cores have their own private first level cache (L1) while they share their second level cache (L2). The first level cache on both the cores is 2-way set associative with cache line size of 2K bytes, and access latency of 30ns per word, while the shared cache is direct mapped with cache line size of 4K bytes and access latency of 80ns per word. Consider a process with two threads running on these cores as follows (assume the size of an integer to be 4 bytes which is same as the word size):

Thread 1:

```
int A[1024];
for (i=0; i < 1024; i++)
{
A[i]  = A[i] + 1;
}
```

Thread 2:

```
int B[1024];
for (i=0; i< 1024; i++)
{
B[i] = B[i] + 1;
}
```

Initially assume that both the arrays A and B are in main memory, whose access latency is 200ns per word. Assume that an int is word sized. Furthermore, assume that A and B when mapped to L2 start at address 0 of a cache line. Assume a write back policy for both L1 and L2 caches.

(a) If the main memory blocks having arrays A and B map to different L2 cache lines, how much time would it take the process to complete its execution in the worst case? (Assuming this is the only process running on the machine.)

(b) If the main memory blocks having arrays A and B map to the same L2 cache line, how much time would it take the process to complete its execution in the worst case? (Assuming this is the only process running on the machine.)

In the worst case, thread 1 could access A[0], thread 2 could access B[0], then thread 1 could access A[1] followed by B[1] access by thread 2 and so on. Every time A[I] or B[i] is accessed, it evicts the other array from L2 cache and so a subsequent access to the other array has to again cause a main memory access.