UCLA CCLE

Winter 2021 - COM SCIM151B-1 / EC ENGRM116C-1 - REINMAN

Started on	Wednesday, 17 March 2021, 8:00 AM PDT
State	Finished
Completed on	Wednesday, 17 March 2021, 10:49 AM PDT
Time taken	2 hours 49 mins
Grade	**90.00** out of 100.00

Question **1**

Correct

2.00 points out of 2.00

I am increasing the size of the cache used in my processor design to reduce the miss rate. I expect that the cycle time of my new processor design

a. could increase due to the larger cache latency ✔

b. must say the same

c. could decrease due to the reduction in miss cycles

Correct

14.00 points out of 14.00

Consider the 5-stage scalar pipeline we covered in class. Assume we have forwarding to deal with data hazards, and branch prediction to deal with control hazards - just like we've covered in class.

The instruction mix for a particular application is:

Instruction	Frequency
R-Type	50%
LW	25%
SW	10%
BEQ/BNE	15%

25% of LW instructions are immediately followed by a dependent instruction. 20% of LW instructions are followed by an independent instruction which is then followed by an instruction that is dependent on the LW. 10% of LW instructions are followed by two independent instructions that are then followed by an instruction that is dependent on the LW. For example: lw $t0, 0($s0) followed by add $t1, $t2, $t3 followed by and $t4, $t5, $t6 followed by or $t0, $t7, $t0 --- where the add/and instructions are independent of the lw and the or instruction is dependent on the lw.

The branch predictor is 80% accurate. Branches are resolved in the EX stage.

There is an instruction cache and data cache for the processor - the instruction cache has a 25% miss rate and the data cache has a 30% miss rate. There is a unified L2 cache that has a 10% miss rate and an access latency of 20 cycles. Main memory has an access latency of 150 cycles. Store misses do not stall the processor pipeline.

First - find the TCPI of this architecture - we will break it into pieces for partial credit purposes - make sure you show your work on the uploaded pages:

BCPI =

1.12

✔

MCPI =

11.38

✔

TCPI =

12.50

✔

means our new pipeline will have 7 stages: IF1 IF2 ID EX MEM1 MEM2 WB --- where the I$ is accessed in IF1 and IF2, and the D$ is accessed in MEM1 and MEM2. We must wait for both MEM stages (MEM1 and MEM2) to get any value out of memory with a load instruction.

The instruction cache miss rate drops to 10% and the data cache miss rate drops to 15%.

Consider how these changes will affect hazards, and compute TCPI again:

BCPI =

1.27

✔

MCPI =

4.81

✔

TCPI =

6.08

✔

Correct

15.00 points out of 15.00

Suppose we have a multicore system (e.g. a CMP as we discussed in class) with two cores (core 0 and core 1). Each core has an L1 data cache which is writethrough and write around. The cores share a single L2 cache, which is writeback and write allocate. The cores use snoopy coherence to maintain cache coherence.

Values here are shown in hex (flashback to CS33). Data in MIPS is arranged in big endian form. If you forgot what that means (your CS33 instructor will be so sad): for example, in Core 0's data cache - address 0x508 holds byte 0xDE, address 0x509 holds byte 0xAD, address 0x50A holds byte 0xBE, and address 0x50B holds byte 0xEF. If you did a load word on address 0x508, you would get the value 0xDEADBEEF.

Here are the relevant initial conditions for the cores:

Core 0

L1 Data Cache:

Valid Bit	Block Address	Data
1	0x500	0x40 00 00 00 80 00 00 00
1	0x508	0xDE AD BE EF DE FE C8 ED

Selected Register Contents:
register $t0 = 0x500
register $t1 = 0xCCCCCCCC

Core 1

L1 Data Cache:

Valid Bit	Block Address	Data
0	0x500	0x30 00 00 00 50 00 00 00
0	0x508	0xFE ED FA CE FE ED FA CE

Selected Register Contents:
register $t0 = 0x504
register $t1 = 0x77777777

Shared L2 Cache

Valid Bit	Dirty Bit	Block Address	Data
1	0	0x500	0x40 00 00 00 80 00 00 00
1	0	0x508	0xDE AD BE EF DE FE C8 ED

Memory

Address	Data
0x500	0x04
0x501	0x00
0x502	0x00
0x503	0x00
0x504	0x08

0x507	0x00
0x508	0xDE
0x509	0xAD
0x50A	0xBE
0x50B	0xEF
0x50C	0xDE
0x50D	0xFE
0x50E	0xC8
0x50F	0xED

Given all of these initial conditions, we are going to evaluate what happens when the following instruction is executed on Core 1:

sw $t1, 8($t0)

This is the only instruction executed by Core 1 or Core 0. Your job is to figure out what the state of the memory hierarchy would be after this instruction is executed:

Core 0
L1 Data Cache

Valid Bit	Block Address	Data
1 ✔	0x500	0x40 00 00 00 80 00 00 00 ✔
0 ✔	0x508	0xDE AD BE EF DE FE C8 ED ✔

Core 1
L1 Data Cache:

Valid Bit	Block Address	Data
0 ✔	0x500	0x30 00 00 00 50 00 00 00 ✔
0 ✔	0x508	0xFE ED FA CE FE ED FA CE ✔

Shared L2 Cache

Valid Bit	Dirty Bit	Block Address	Data
1 ✔	0 ✔	0x500	0x40 00 00 00 80 00 00 00 ✔
1 ✔	1 ✔	0x508	0xDE AD BE EF 77 77 77 77 ✔

Memory

Does memory (e.g. physical memory) change?

No ✔

UCLA CCLE

Question **4**

Incorrect

0.00 points out of 2.00

My compiler is having a difficult time with register file coloring, which is leading to increased register spilling in my code. Which of the following optimizations would not be targeted towards reducing the impact of that spilling?

a. out of order instruction execution with register renaming

b. increasing the size of the register file

c. branch delay slots

d. increasing the size of the data cache ✖

Correct

18.00 points out of 18.00

Suppose we have a memory system with 32-bit virtual addresses (so 2^{32}B of virtual memory) and 31-bit physical addresses (so 2^{31}B of physical memory). Our memory uses 2^{11}B pages. We have a 64B L1 data cache that is VIPT and direct mapped, and has a 16B block size. We have a 512B L2 cache that is PIPT and 4-way set associative, and has a 16B block size. The L2 cache uses LRU replacement. There is a TLB to help with translation - and the TLB is fully associative and can hold 16 translations.

Assume that the caches and TLB are initially empty.

Assume that there are no page faults - all relevant pages of virtual memory that are addressed below are already mapped to unique physical pages in memory.

Given this information, fill in the hit/miss behavior for the table below:

Virtual Address	L1 Hit?	L2 Hit?	TLB Hit?
...0001011011100	Compulsory	Miss	Miss
...0001010010000	Compulsory	Miss	Hit
...0111010011000	Compulsory	Miss	Miss
...0001010011000	Conflict	Hit	Hit
...0000010011100	Compulsory	Miss	Hit
...0101010010000	Compulsory	Miss	Hit
...0001011011100	Capacity	Hit	Hit
...0001010010000	Conflict	Hit	Hit
...0111010011000	Capacity	Hit	Hit
...0001010011000	Conflict	Hit	Hit
...0000010011100	Capacity	Hit	Hit
...0101010010000	Capacity	Hit	Hit

Assume that all leading bits of the address (e.g. the "...") are 0's. Even if the L1 hits, still note whether the L2 would have been a hit or a miss. For L1 misses, specify the type of miss (compulsory, conflict, capacity).

Incorrect

0.00 points out of 2.00

A new processor generation has instructions that can do a multiply and accumulate instruction, instead of doing separate multiply and add instructions. The multiply and accumulate instruction takes 10 cycles to execute, the separate multiply instruction takes 7 cycles, and the separate add instruction takes 4 cycles. I recompile my code to take advantage of this new instruction. I would expect that my cycles per instruction

a. could increase due to the complexity of the new instructions

b. could decrease due to the efficiency of the new instructions ✖

c. must stay the same

Correct

12.00 points out of 12.00

Consider the following code:

HERE: lw $t0, 8($s0)

add $t0, $t0, $s1

lw $t0, 0 ($t0)

add $t9, $t9, $t0

addi $s0, $s0, 16

bne $s0, $s2, HERE

We are going to statically schedule this code for a 2 wide superscalar processor - just like the one we covered in class. Branches will be resolved in the EX stage, and we will use branch delay slots to deal with control hazards. We will make use of forwarding to resolve our data hazards, except for the load use hazard that we had in our 5 stage pipeline from class. Like the 2 wide superscalar processor we covered in class, we will specialize our datapath so that we may only execute one ALU/BR instruction and one LW/SW instruction each cycle.

Without doing any unrolling - first schedule the code above in the slots below. To help you, we have scheduled the addi for you already. For the remaining instructions, select them from the dropdown below. There should only be one unique answer with the given placement of the addi instruction. If no instruction fits in a slot, use a nop from the dropdown.

VLIW Schedule (no Unrolling)

LW/SW	ALU/BRANCH
lw $t0, 8($s0) ✔	nop ✔
nop ✔	addi $s0, $s0, 16
nop ✔	add $t0, $t0, $s1 ✔
lw $t0, 0 ($t0) ✔	bne $s0, $s2, HERE ✔
nop ✔	nop ✔
nop ✔	add $t9, $t9, $t0 ✔

Now take the code and unroll it once to make two copies of the loop body. We will use register $t1 to rename. Schedule it below - again, we have placed the addi for you.

VLIW Schedule (Unroll)

LW/SW	ALU/BRANCH
lw $t0, 8($s0) ✔	nop ✔
lw $t1, 24($s0) ✔	addi $s0, $s0, 32

nop ✔	add $t0, $t0, $s1 ✔
lw $t0, 0 ($t0) ✔	add $t1, $t1, $s1 ✔
lw $t1, 0 ($t1) ✔	bne $s0, $s2, HERE ✔
nop ✔	add $t9, $t9, $t0 ✔
nop ✔	add $t9, $t9, $t1 ✔

Question **8**

Correct

2.00 points out of 2.00

My algorithm has been changed from bubble sort (yay) to quicksort, for the same processor and same compiler. I would expect that the cycle time of my processor:

a. could increase due to the complexity of the algorithm

b. must stay the same ✔

c. could decrease due to the efficiency of the algorithm

Question **9**

Incorrect

0.00 points out of 2.00

If your program frequently has stores that write to memory locations that are never read by load instructions, and has load instructions that frequently miss and are critical to program performance (e.g. you don't want to add any latency to the miss), you would expect that the best cache design for that pattern of accessories would be:

a. writethrough and write around ✖

b. writeback and write around

c. writeback and write allocate

d. writethrough and write allocate

Correct

12.00 points out of 12.00

Consider the following program fragment:

HERE: lw $t0, 0 ($t1)

add $t2, $t3, $t4

lw $t3, 0 ($t2)

sub $t4, $t3, $t0

beq $t4, $t5, THERE

lw $t0, 4 ($s0)

add $s0, $s0, $t0

THERE: add $t1, $t1, $s2

bne $t1, $s1, HERE

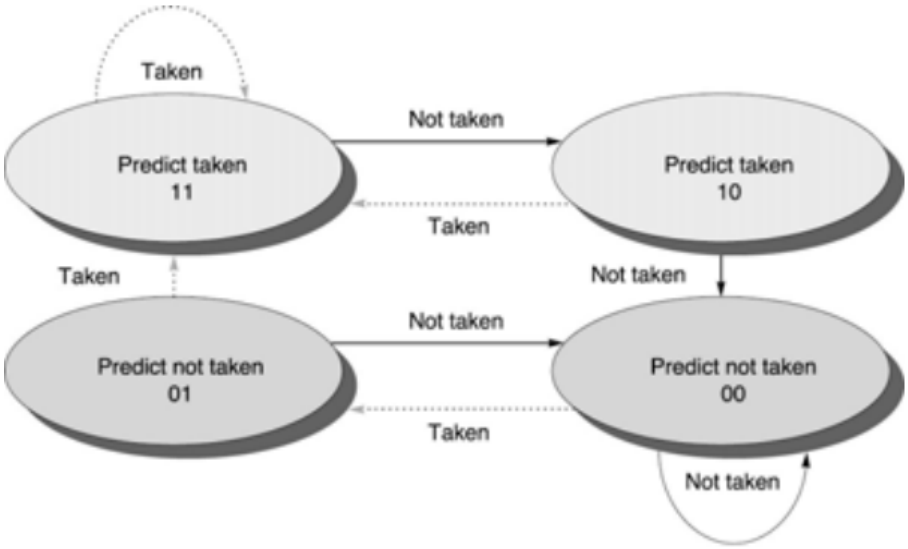
These instructions are stored contiguously in memory.

The dynamic behavior of the branches is as follows:

The BEQ is taken, then not taken, and then taken.

The BNE is taken, then taken, and then not taken.

We are going to use a 2-bit branch predictor to handle control hazards for this sequence of instructions. The 2-bit branch predictor will have four entries (each with a 2-bit counter). The following FSM defines the behavior of the 2-bit counter:



Each branch predictor entry is at the 01 state (predict not taken) at the start of the code fragment.

We will index the branch predictor via the same hash we used in class: **(PC >> 2) % (table size)**

For each dynamic instance of the branch instructions, indicate whether the branch predictor was correct or incorrect:

BEQ	Incorrect	Incorrect	Correct
BNE	Correct	Correct	Incorrect

Question 11

Correct

2.00 points out of 2.00

I am modifying our five stage type line from class to a six stage pipeline by implementing a staggered ALU. Which of the following effects would I not expect from the implementation?

a. An increase in the frequency of control hazards

b. An increase in the frequency of data hazards

c. A decrease in forwarding complexity

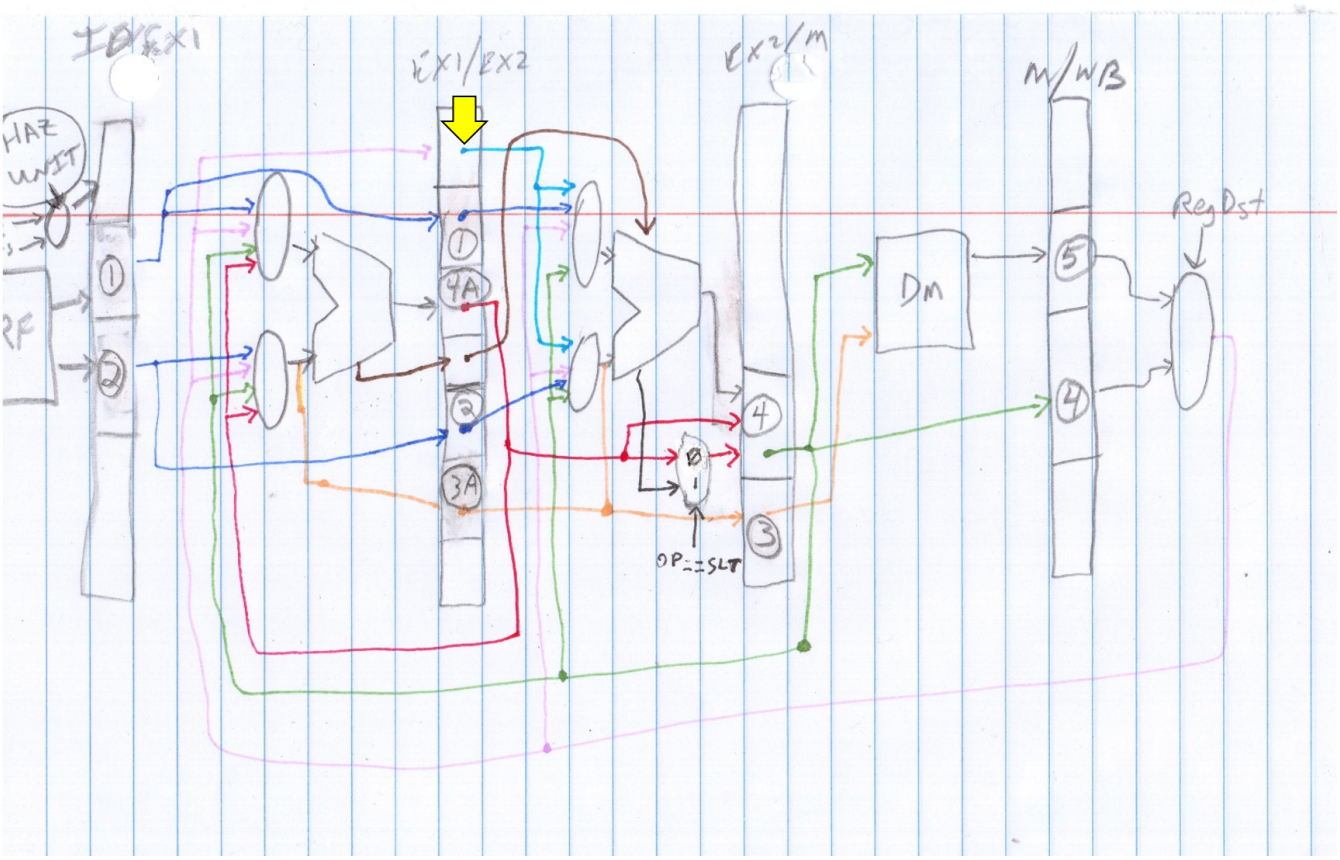
d. A decrease in cycle time

UCLA CCLE

Correct

2.00 points out of 2.00

Here is the staggered ALU that we covered in class. I've added a yellow arrow at the top of the EX1/EX2 latch. The yellow arrow is pointing to a light blue wire. If instruction A is currently in the EX2 stage, the data carried on that blue wire can best be described by which of the following:



- a. the result of the instruction that was in WB in the last cycle ✔
- b. the result of the instruction that was in EX1 in the last cycle
- c. the result of the instruction currently in MEM
- d. the result of the instruction currently in WB
- e. the result of the instruction currently in EX2
- f. the result of the instruction currently in EX1
- g. the result of the instruction that was in EX2 in the last cycle
- h. the result of the instruction that was in MEM in the last cycle

UCLA CCLE

Question 13

Incorrect

0.00 points out of 2.00

I have a five stage pipelined processor using branch delay slots to deal with control hazards. I am redesigning the entire system to make use of branch prediction. What impact could I expect to see from the change?

a. A decrease in instruction count

b. A decrease in data cache misses

c. A decrease in pipeline flushes ✖

d. A decrease in cycle time

Question 14

Incorrect

0.00 points out of 2.00

We will compare a scalar pipelined processor to a single cycle processor. Which of the following would be an advantage of the pipeline processor?

a. hazards

b. design complexity

c. instructions per cycle ✖

d. cycle time

Correct

2.00 points out of 2.00

Suppose you are creating the next generation processor architecture for an existing system. The current generation has a large number of capacity misses in the level one data cache. Assume that the total size of the cache in the next generation will stay the same as the current generation. Which of the following design options would definitely NOT lead to a reduction in those misses?

a. increasing the block size

b. increasing the number of indices (sets)

c. changing the eviction policy

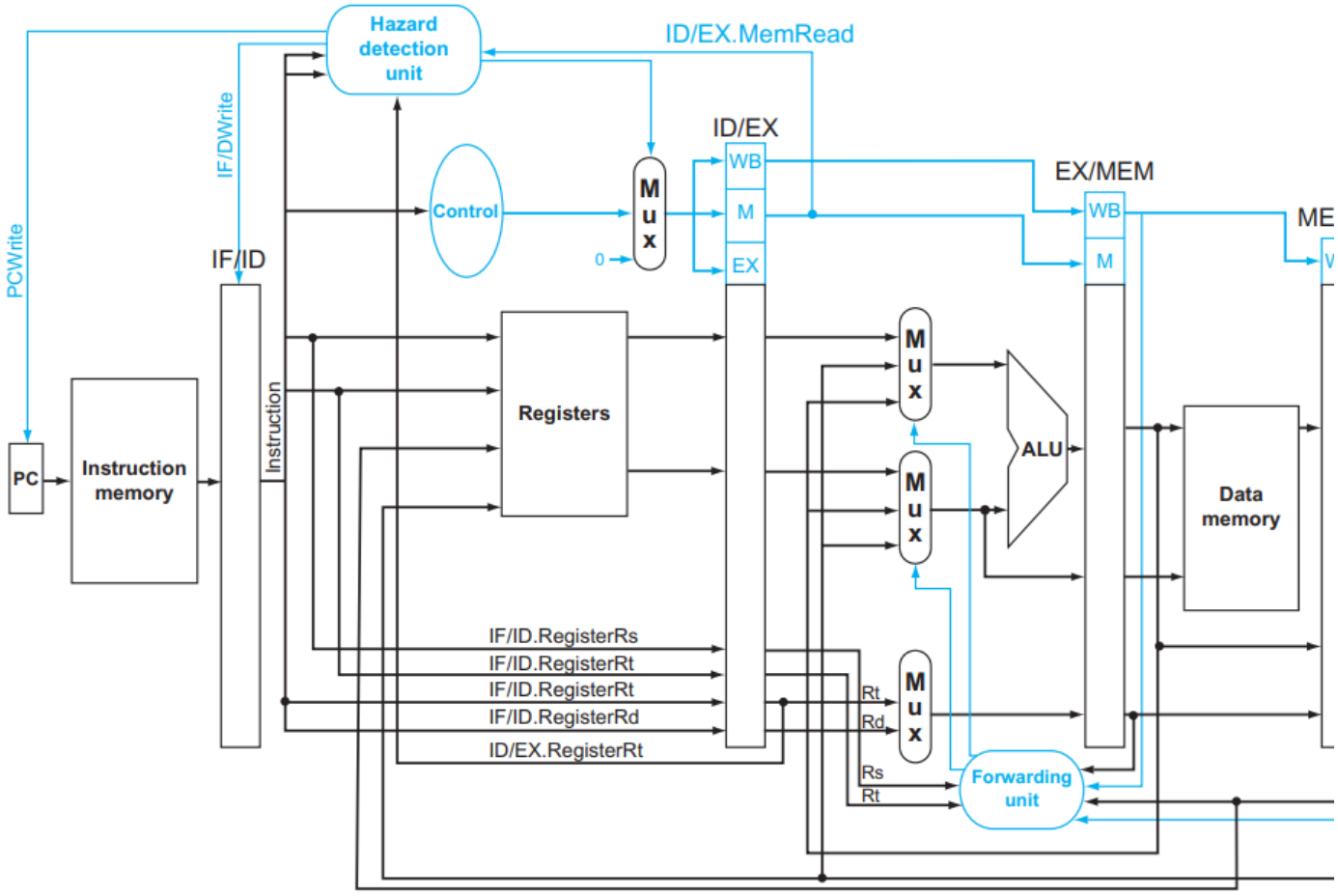
d. increasing the associativity ✔

UCLA CCLE

Correct

9.00 points out of 9.00

Consider the 5-stage pipeline we covered in class, with the implementation of forwarding logic. Here's the design from your textbook:



Note that there are two units that handle data hazards here - the hazard detection unit and the forwarding unit.

When we implemented forwarding logic for that pipeline, we used the following control for the internals of the forwarding logic related to the EX hazard portion of the forward:

EX hazard:

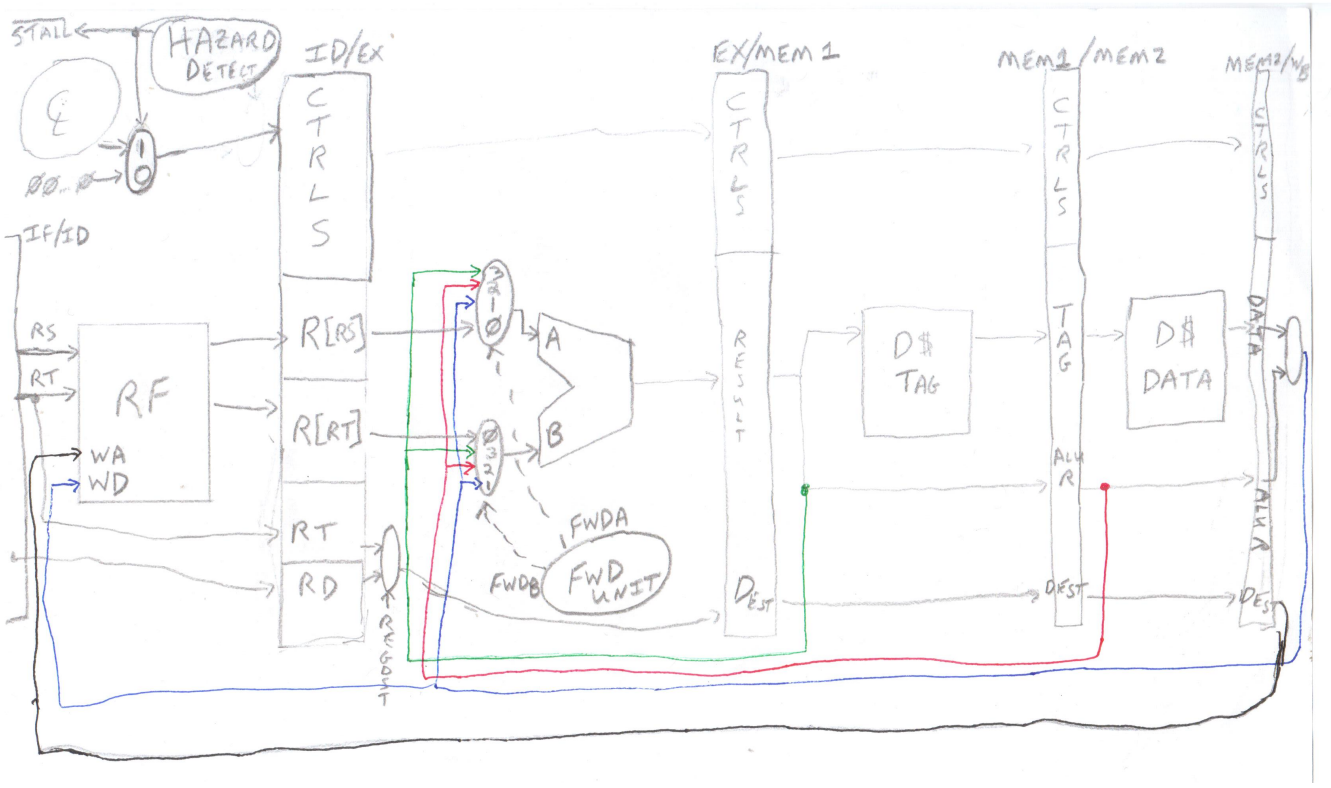
```
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10

if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB = 10
```

And the following control for the internals of the hazard detection unit logic:

```
((ID/EX.RegisterRt = IF/ID.RegisterRs) or
  (ID/EX.RegisterRt = IF/ID.RegisterRt)))
  stall the pipeline
```

We are going to extend the pipeline to 6 stages - separating the memory stage into Mem1 and Mem2 as shown in this professional diagram below:



Note that there are two forwarding muxes in front of the ALU, just like in the 5 stage pipeline, and these are controlled by the forwarding unit (FWD UNIT on the diagram above) via signals FWDA and FWDB. There is also a hazard detection unit (HAZARD DETECT on the diagram) that has output $\overline{\text{STALL}}$. This is the way that we handled the hazard detection unit in class - with a single signal instead of multiple signals as in your textbook's diagram of the 5-stage pipeline.

I have colored the inputs to the forward logic as blue (option 1 - from the instruction in the WB stage), red (option 2 - from the instruction in the MEM2 stage), green (option 3- from the instruction in the MEM stage), and grey (option 0 - the original value from the register file read).

Your job is to come up with new forwarding control logic and hazard detection logic. We will give you a starting point, and you will need to fill in the blanks to complete it.

FWD UNIT (we'll just do the EX hazard portion - e.g. forwarding from the instruction in the EX stage):

EX hazard:

if (EX/MEM1. RegWrite ✔ and (EX/MEM1.RegisterRd ≠ 0) and (EX/MEM1.RegisterRd = ID/EX. RegisterRs ✔)) FWDA = 11 ✔

11 ✔

HAZARD DETECT:

if ((ID/EX.MemRead and ((ID/EX.RegisterRt = IF/ID.RegisterRs) or (ID/EX.RegisterRt = IF/ID.RegisterRt))) or (EX/MEM1 ✔ .MemRead and ((EX/MEM1 ✔ .RegisterRt = IF/ID.RegisterRs) or (EX/MEM1 ✔ .RegisterRt = IF/ID.RegisterRt)))) stall the pipeline

◄ Final Exam Choice

Jump to...

Final Exam Work Submission ►