

1. You are to assume the same 5-stage pipeline discussed in class when answering these questions.

Assume that the first register in an arithmetic operation is the destination register, e.g., in "ADD R3, R2, R1" register R3 receives the result of adding registers R2 and R1.

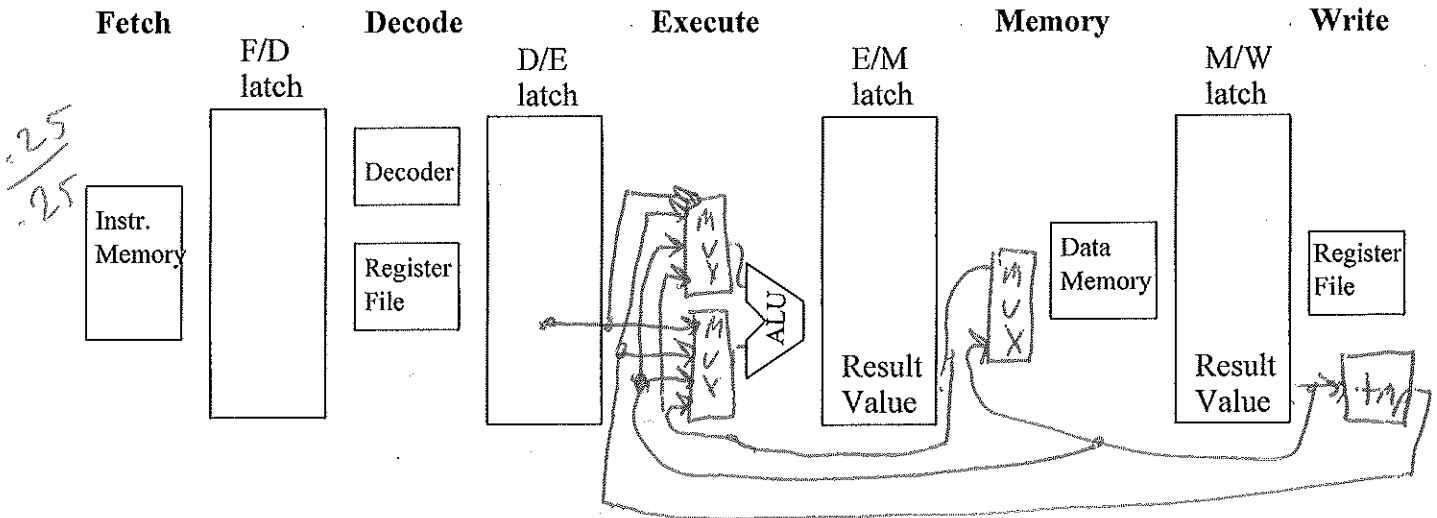
a. What would the timing be **without** bypass-signal paths/forwarding (use "stalls" to solve the data hazard)? (This code might require more or less that 15 cycles)

	Time →																					
Instructions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
ADD R3, R2, R1	F	D	E	M	W																	
LOAD R4, 16(R3)		F	-	-	-	D	E	M	W													
STORE R4, 8(R3)						F	-	-	-	D	E	M	W									
SUB R3, R4, R1										F	D	E	M	W								
MUL R6, R3, R4											F	-	-	-	D	E	M	W				
STORE R6, 4(R5)															F	-	-	-	D	E	M	W

b. What would the timing be **with** bypass-signal paths? (This code might require more that 15 cycles)

	Time →														
Instructions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ADD R3, R2, R1	F	D	E	M	W										
LOAD R4, 16(R3)		F	D	E	M	W									
STORE R4, 8(R3)			F	D	E	M	W								
SUB R3, R4, R1				F	D	E	M	W							
MUL R6, R3, R4					F	D	E	M	W						
STORE R6, 4(R5)						F	D	E	M	W					

c. Draw ALL the bypass-signal paths needed for the above example.



2. Suppose that you are writing a compiler for a machine that has opcodes to statically predict whether or not branches will be taken (BEQ, BEQ_PREDICT_TAKEN, BEQ_PREDICT_NOT_TAKEN, etc.). For each of the following HLL statements, predict whether or not the compiler should predict taken or not. (Briefly justify your answer)

- | | | | |
|---------------------------------|---------------------------------|---|---|
| a) integer x
if (x > 0) then | b) integer x
if (x = 0) then | c) integer i
for i := 1 to 500 do | d) char ch
if (ch >= 'a' and ch <= 'z') then |
| end if | end if | end for | end if |
| NOT_TAKEN
x mostly positive | TAKEN x rarely
zero | NOT_TAKEN only
take once when drop out | Either accepted depending on
justification |

3. Consider the following bubble sort algorithm that sorts an array numbers[1..n]:

```
BubbleSort (int n, int numbers[])
```

```

int bottom, test, temp;
boolean exchanged = true;
bottom = n - 2;
while (exchanged) do
    exchanged = false;
    for test = 0 to bottom do
        if number[test] > number[test + 1] then
            temp = number[test];
            number[test] = number[test + 1];
            number[test + 1] = temp;
            exchanged = true;
        end if
    end for
    bottom = bottom - 1;
end while
end BubbleSort

```

	Part (a) answer	Part (b) answer
	conditional	Predict NOT_TAKEN since loop
	conditional	Predict NOT_TAKEN since loop
	conditional	No good prediction comparing adjacent array locations either outcome = likely
	unconditional	
	unconditional	

- 25 a) Where in the code would unconditional branches be used and where would conditional branches be used?
- 25 b) If the compiler could predict by opcode for the conditional branches (i.e., select whether to use machine language statements like: "BRANCH_LE_PREDICT_NOT_TAKEN" or "BRANCH_LE_PREDICT_TAKEN"), then which conditional branches would be "PREDICT_NOT_TAKEN" and which would be "PREDICT_TAKEN"?

- c) Assumptions:
- n = 100 and the numbers are initially in descending order before the bubble sort algorithm is called
 - the five-stage pipeline of the lecture
 - the outcome of conditional branches are known at the end of the E stage
 - target addresses of all branches are known at the end of the D stage
 - ignore any data hazards

Under the above assumptions, answer the following questions:

i) If fixed predict-never-taken is used by the hardware, then what will be the total branch penalty (# cycles wasted) for the algorithm? (Here assume NO branch-prediction buffer)

while - cond	for - cond	if - cond	B FOR uncond	B WHILE uncond
1 * 2 (dropping out)	99 * 2 (drop out of loop 99 times)	0 (always true so never fetch wrong)	1 * (99 + 98 + ... + 1) = 4950	99 * 1

Total = 5249 cycles

ii) If a branch-prediction buffer with one history bit per entry is used, then what will be the total branch penalty (# cycles wasted) for the algorithm? (Assume predict-not taken is used if there is no match in the branch-prediction buffer) Explain your answer.

while - cond	for - cond	if - cond	B FOR uncond	B WHILE uncond
1 * 2 (dropping out)	2 + 4*98 (wrong first & last iteration)	0 (always true so never fetch wrong)	1 on first encounter	1 on first encounter

Total = 398 cycles

iii) If a branch-prediction buffer with two history bits (i.e., wrong twice before changing prediction) per entry is used, then what will be the total branch penalty (# cycles wasted) for the algorithm? (Assume predict-not taken is used if there is no match in the branch-prediction buffer) Explain your answer.

while - cond	for - cond	if - cond	B FOR uncond	B WHILE uncond
1 * 2 (dropping out)	99 * 2 (wrong last iteration)	0 (always true so never fetch wrong)	1 on first encounter	1 on first encounter

Total = 202 cycles

4