Mark F.

Comp. Arch. HW #3 Due: Friday, Sept. 26 by 4 PM

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)

.5

	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 R1, 8(R4)						F	–	–	–	D	E	M	W		
SUB R3, R4, R1										F	D	E	M	W	
MUL R6, R3, R4											F	–	–	–	D
STORE R6, 4(R5)															F

16 17 1
EMW
- - -
M 20 2
D E M

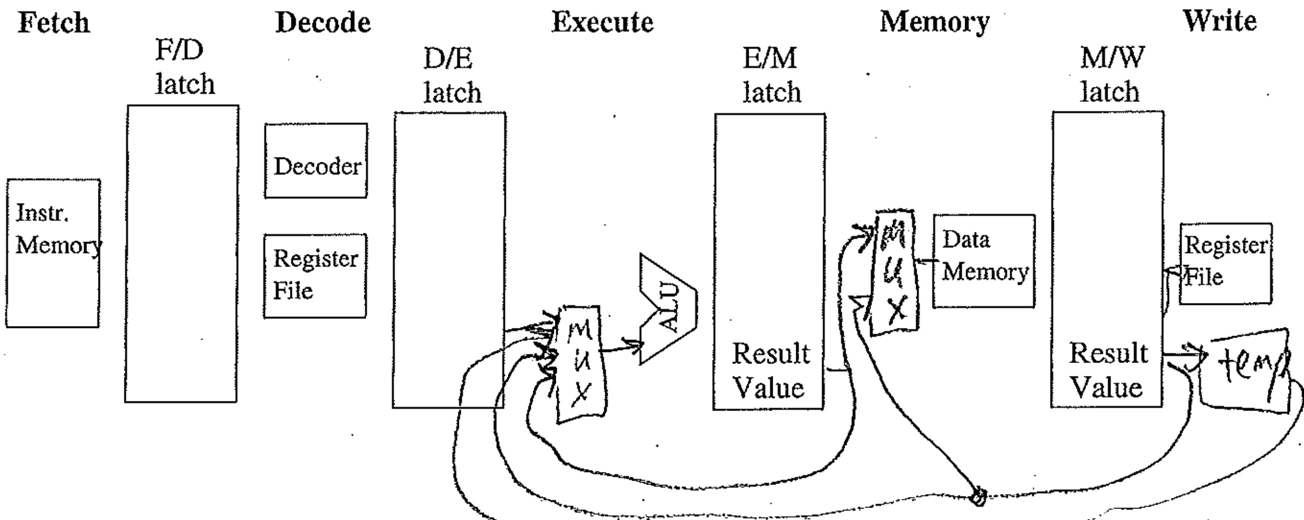
b. What would the timing be **with** bypass-signal paths? (This code might require more that 15 cycles)

.5

	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 R1, 8(R4)			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 and MUXs needed for the above example.

25



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)

1

a) integer x
if (x > 0) then

end if

Predict NOT TAKEN since most common integer values are positive

b) integer x
if (x = 0) then

end if

Predict TAKEN since only one integer value is zero

c) integer i
for i := 1 to 500 do

end for

Predict NOT TAKEN since we loop 500 times and

d) char ch
if (ch >= 'a' and ch <= 'z') then

end if

Hard to predict!

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

BubbleSort (int n, int numbers[])	Part (a) answer	Part (b) answer
int bottom, test, temp;	.25	.25
boolean exchanged = true;		
bottom = n - 2;		
while (exchanged) do	conditional branch	Predict NOT-TAKEN
exchanged = false;		
for test = 0 to bottom do	conditional branch	Predict NOT-TAKEN
if number[test] > number[test + 1] then	Conditional branch	- no good prediction
temp = number[test];		
number[test] = number[test + 1];		
number[test + 1] = temp;		
exchanged = true;		
end if		
end for	uncond. branch	
bottom = bottom - 1;		
end while	uncond. branch	
end BubbleSort		

a) Where in the code would unconditional branches be used and where would conditional branches be used?

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)

.5 — while cond: 2 (drop out); for cond: 2×99; if cond: 0 (never wrong); (end for) jump for: (99+98+...+1)×1 = 100*(99/2) = 4950; (end while) jump while: 99×1; (= 5240)

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.

.5 — while: 2; for: 2+4×98; if: 0; (end for) jump for: 1; (end while) jump while: 1; = (396)

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.

.25 / 1.75 — while: 2; for: 2×99; if: 0; jump for: 1; jump while: 1; = (202)

(4)