HW #7 : 6.1, 6.3, 6.6, 6.10 Matt Kwong

6.1 [M] Consider the following instructions at the given addresses in the memory:

A: 10

1000	Add	R3, R2, #20
1004	Subtract	R5, R4, #3
1008	And	R6, R4, #0x3A → 00111010
1012	Add	R7, R2, R4

Initially, registers R2 and R4 contain 2000 and 50, respectively. These instructions are executed in a computer that has a five-stage pipeline as shown in Figure 6.2. The first instruction is fetched in clock cycle 1, and the remaining instructions are fetched in successive cycles.

(*a*) Draw a diagram similar to Figure 6.1 that represents the flow of the instructions through the pipeline. Describe the operation being performed by each pipeline stage during clock cycles 1 through 8.

(*b*) With reference to Figures 5.8 and 5.9, describe the contents of registers IR, PC, RA, RB, RY, and RZ in the pipeline during cycles 2 to 8.

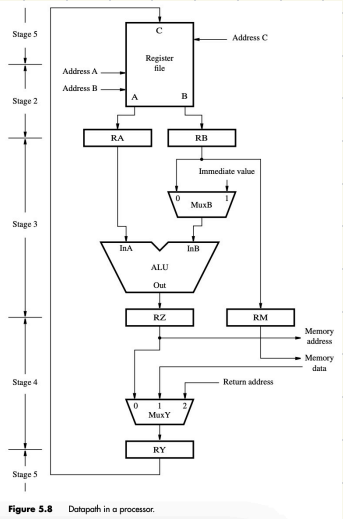


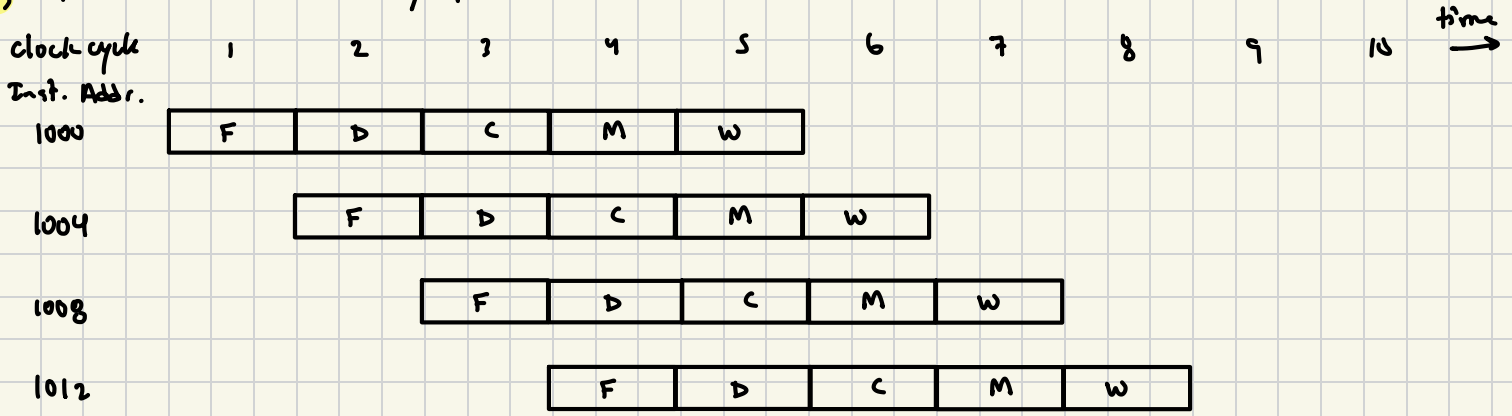
Figure 5.8 Datapath in a processor.

Initially
R2 : 2000
R4 : 50

5-stage processor
1. fetch : F
2. decode : D
3. compute : C
4. memory : M
5. write : W

No data dependencies:
- R4 used multiple times, BUT for read only.
No stalls from computing dependencies

a) Flow of instructions through pipeline



clock cycle / Inst. Addr.	1	2	3	4	5	6	7	8
1000 ADD R3, R2, #20	IR ← [1000]	RA ← [R2] RB ← [R3]	RZ ← [R2] + 20	RY ← [RZ]	R3 ← [RY]			
1004 SUBTRACT R5, R4, #3		IR ← [1004]	RA ← [R4] RB ← [R5]	RZ ← [R4] - 3	RY ← [RZ]	R5 ← [RY]		
1008 AND R6, R4, #0x3A			IR ← [1008]	RA ← [R4] RB ← [R6]	RZ ← [RY] • 0x3A	RY ← [RZ]	R6 ← [RY]	
1012 ADD R7, R2, R4				IR ← [1012]	RA ← [R2] RB ← [R4]	RZ ← [R2] + [R4]	RY ← [RZ]	R7 ← [RY]

R4 • 0x3A

↓ ↓

bitwise $(50)_{10} \times (3A)_{16}$ =

00110010
× 00111010
00110010

$2^5 + 2^4 + 2^1$
$= 32 + 16 + 2$
$= (50)_{10}$

Matt Krueger

6.1 [M] Consider the following instructions at the given addresses in the memory:

RA RB

1000	Add	R3, R2, #20
1004	Subtract	R5, R4, #3
1008	And	R6, R4, #0x3A
1012	Add	R7, R2, R4

Initially, registers R2 and R4 contain 2000 and 50, respectively. These instructions are executed in a computer that has a five-stage pipeline as shown in Figure 6.2. The first instruction is fetched in clock cycle 1, and the remaining instructions are fetched in successive cycles.

(*a*) Draw a diagram similar to Figure 6.1 that represents the flow of the instructions through the pipeline. Describe the operation being performed by each pipeline stage during clock cycles 1 through 8.

(*b*) With reference to Figures 5.8 and 5.9, describe the contents of registers IR, PC, RA, RB, RY, and RZ in the pipeline during cycles 2 to 8.

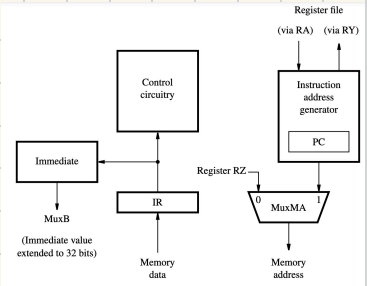


Figure 5.9 Instruction fetch section of Figure 5.7.

b)

clock cycle / Register	1	2	3	4	5	6	7	8
IR	1000	1004	1008	1012	1016	...	...	
PC (incremented during fetch stage)	1004	1008	1012	1016	+4	+4	...	...
RA		2000 (R2)	50 (R4)	50 (R4)	2000 (R2)			
RB		R3 (R3)	R5 (R5)	R6 (R6)	50 (R4)			
RZ			2020 (RA+20)	47 (RA-3)	50 (50 & 58 = 50)	2050 (RA+R4)		
RY				2020 (RZ from prev)	47 (RZ from prev)	50 (RZ from prev)	2050 (RZ from prev)	

Matt Kemp

6.3 [M] Consider the loop in the program of Figure 2.8. Assume it is executed in a five-stage pipeline with forwarding paths to the ALU from registers RY and RZ in Figure 5.8. Assume that the pipeline uses static branch prediction with a not-taken assumption. Draw a diagram similar to Figure 6.1 for the execution of two successive iterations of the loop.

0:	Load	R2, N	Load the size of the list.	
4	Clear	R3	Initialize sum to 0.	
8	Move	R4, #NUM1	Get address of the first number.	
12	LOOP: Load	R5, (R4)	Get the next number.	] *1
16	Add	R3, R3, R5	Add this number to sum.	
20	Add	R4, R4, #4	Increment the pointer to the list.	
24	Subtract	R2, R2, #1	Decrement the counter.	
28	Branch_if_[R2]>0	LOOP	Branch back if not finished.	
32	Store	R3, SUM	Store the final sum.	] *2
	...			

Figure 2.8 Use of indirect addressing in the program of Figure 2.6.

— PLEASE READ ASSUMPTIONS —

1. instructions start @ 0
2. optimized compiler:

- if compiler is not optimized there would be stalls at *1 and *2 Due to LOAD/STORE operations; accessing memory is costly

To reduce LOAD stalls & dependencies: swap orders:

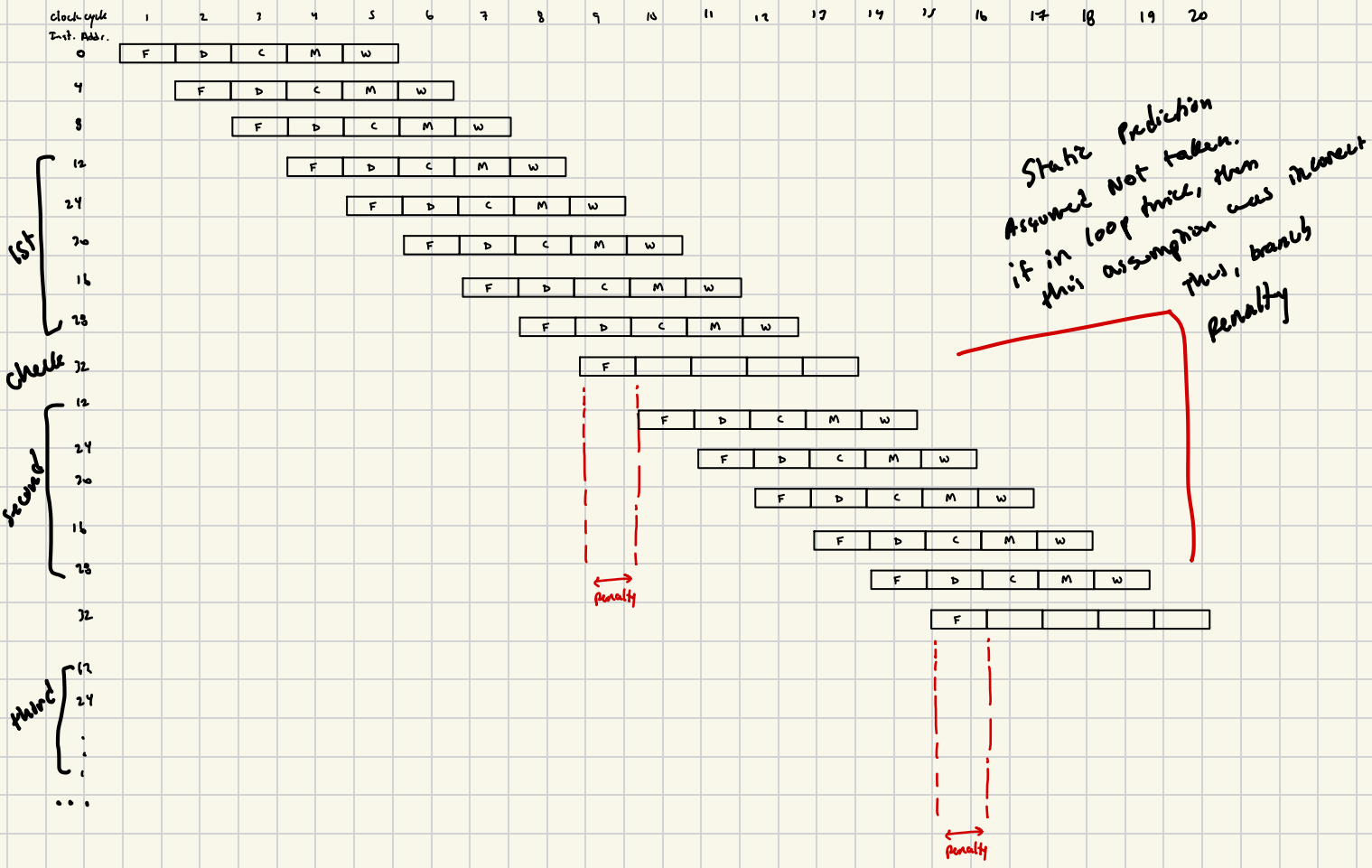
16 ADD R3, R3, R5 (dependent on previous inst I_{12})

&

24 SUBTRACT R2, R2, #1 (following instruction I_{28} dependant)

* optimized compiler will put "useful" command directly after three commands to avoid 1 cycle delay

With this "more useful" order, stalls are mitigated & data forwarding avoided.



Matt Krueger

6.6 [M] The forwarding path in Figure 6.5 allows the contents of register RZ to be used directly in an ALU operation. The result of that operation is stored in register RZ, replacing its previous contents. This problem involves tracing the contents of register RZ over multiple cycles. Consider the two instructions

I_0 : ALU RZ = 17

I_1:	Add	R3, R2, R1
I_2:	LShiftL	R3, R3, #1
I_3:	unknown	

While instruction I_1 is being fetched in cycle 1, a previously fetched instruction is performing an ALU operation that gives a result of 17. Then, while instruction I_1 is being decoded in cycle 2, another previously fetched instruction is performing an ALU operation that gives a result of 198. Also during cycle 2, registers R1, R2, and R3 contain the values 30, 100, and 45, respectively. Using this information, draw a timing diagram that shows the contents of register RZ during cycles 2 to 5.

cycle	1	2	3	4	5	6
RZ	17	198	130	260	?	

cycle 1:

RZ = 17

cycle 2:

R1 = 30
R2 = 100
R3 = 45
RZ = 198

cycle 3:

R3 ← [R2] + [R1] = 30 + 100
RZ ← [R3]
∴ RZ = 130

cycle 4:

Logical shift left: <<1

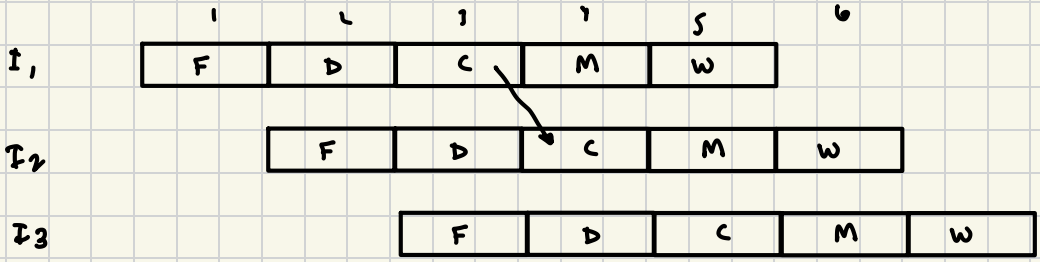
since R3 dependency, RZ is forwarded from ALU compute stage of cycle 3 to ALU compute stage of cycle 4

NO CARRY in LSHIFTL

RZ = 130 × 2
= 260

cycle 5:

unknown; instruction not given



Matt Kruger

6.10 [M] Additional control logic is required in the pipeline to forward the value of register RZ as shown in Figure 6.5. What specific conditions must this additional logic check to determine the settings of the multiplexers feeding the ALU inputs in the Compute stage of the pipeline?

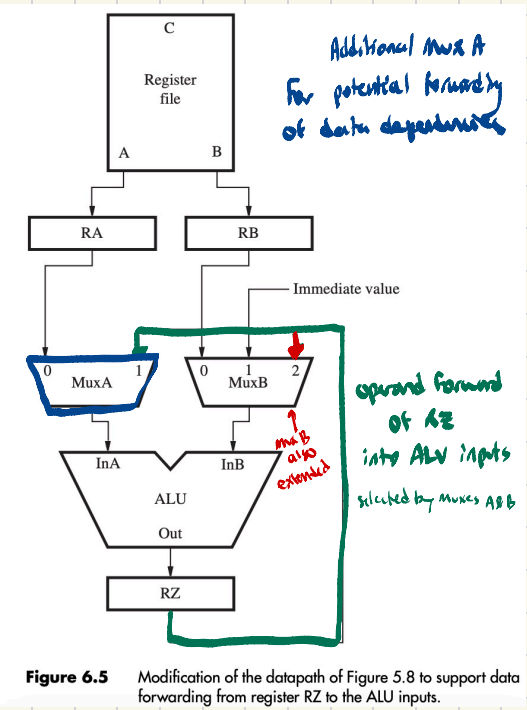


Figure 6.5 Modification of the datapath of Figure 5.8 to support data forwarding from register RZ to the ALU inputs.

muxes must check for dependencies between instructions

- MuxA is added & MuxB extended to allow either
  1. the value read from the register file
     OR
  2. RZ from previous ALU result
     OR
  3. immediate value

Control signal logic:

- The destination of previous ALU result (I_j) is compared with sources of successive ALU compute stage (I_{j+1}).
  This comparison dictates the multiplexer settings for muxes A & B

  I_j result & I_{j+1} source(s) match,
  then RZ is forwarded to next compute stage to avoid stalls.