Pledge: *I pledge my honor that I have abided by the Stevens Honor System.* -Eric Altenburg

4.3: Page 369

(4.3.1)

LDUR and STUR use the data memory resulting in **35%**.

(4.3.2)

All instructions need to be fetched from instruction memory. Therefore, all types of instructions use instruction memory to reside and be fetched from. **100%**.

(4.3.3)

The LDUR and STUR use sign-extension for the address offset. I-type has a 12-bit immediate value that needs to be zero-extended. Compare and Branch use a 19-bit field which needs to be extended to 64-bit address. Therefore, the total fraction of instructions includes **76%**.

(4.3.4)

Although it is computed every cycle, when the sign-extended is not used, it is neglected since only 76% of the total amount of instructions will utilize it.

4.5: Page 369

After converting to binary, the hex becomes:

1111 1000 0000 0001 0100 0000 0110 0010

The op code is 11111000000 (bits 31-21).

This converted to decimal is 1984 which in hex is 7C0.

From this we can conclude that it is a D-type instruction.

000010100 is the offset (bits 20-12).

00 is the op (bits 11-10).

00011 is the Rn (bits 9-5).

00010 is the Rt (bits 4-0).

(4.5.1)

After moving through the sign extended, the output is:

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0001 0100

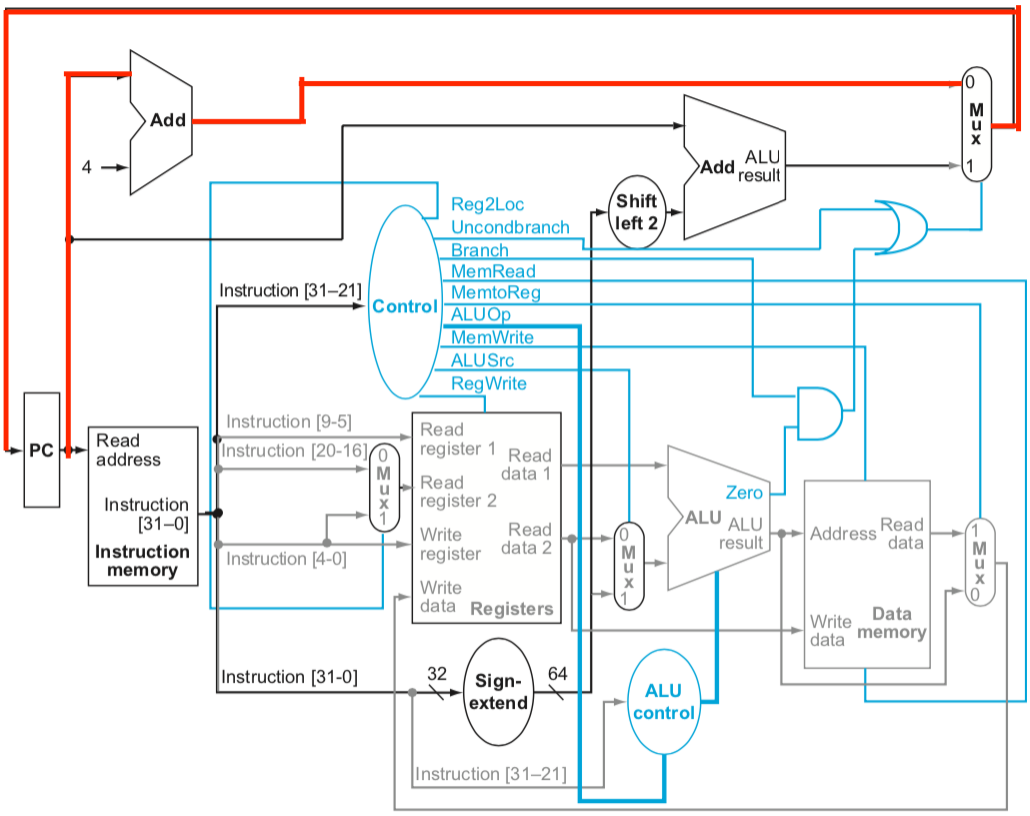
Output of the "shift left 2":

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0101 0000

(4.5.2)

The value of the ALU control unit's input for this instruction is 0010.

(4.5.3)



After following the highlighted path in red, 4 gets added making the new PC address **PC+4**.

4.8: Page 371

Assume for this problem an RType/IType takes 725 ps, LDUR takes 925 ps, STUR takes 880 ps, CBZ takes 735 ps, and B takes 525 ps.

$CPU_{new} = (0.52 * 725) + (0.25 * 925) + (0.1 * 880) + (0.11 * 735) + (0.02 * 525) =$ **787.6 ps**

$CPU_{old} = 930\ ps$

$Speedup = \frac{930}{787.6} = 1.18$ or an increase of **18%**.

4.9: Page 371

For this problem, assume the clock cycles is 925 ps.

(4.9.1)

Without the multiplier, the clock cycle time would be **925 ps**.

With the multiplier, the added 300 ps latency would result in a clock cycle time of **1225 ps**.

(4.9.2)

$Speedup_{new} = \frac{1}{0.95} * \frac{925ps}{1225ps} =$ **79.5%**.

This is not actually a speedup, instead, it is a decrease in performance as the clock cycle time increased after adding in the multiplier.

(4.9.3)

$(925 + x) * 0.95 < 925$

$x < 48.68$ ps

Although the new CPU has 5% fewer instructions, at most the new cycle time can add 48.68 ps to the ALU latency. However, from the problem, it instead adds 300 ps, therefore, the new ALU unit cannot be any slower and still result in an increase in performance.

4.16: Page 373

(4.16.1)

Pipelined: Slowest of the times which is **350 ps**.

Non-Pipelined: Each instruction is completed so adding them up gets us a clock cycle time of **1250 ps**.

(4.16.2)

Pipelined: Since it takes 5 cycles, $5 * 350 =$ **1750 ps**.

Non-pipelined: Just like the previous part, you would add up the times for each stage so **1250 ps**.

(4.16.3)

In choosing which stage to split, one should consider splitting the longest one which will then change the longest stage. The old longest stage, found in 4.16.1, was **350 ps**. The new longest stage after splitting up the aforementioned stage is **MEM at 300 ps**.

(4.16.4)

Since the data portion of the memory is only accessed by LDUR and STUR, the percentage of the clock cycles is only comprised of these two instructions which is **35%**.

4.18: Page 374

CC	1	2	3	4	5	6	7
ADDI X2, X2, #5	IF	ID	EX	ME	WB		
ADD X3, X1, X2		IF	ID	EX	ME	WB	
ADDI X4, X1, #15			IF	ID	EX	ME	WB

X3 = 33

X4 = 26

4.20: Page 374

```
ADDI  X1,  X2,  #5
NOP
NOP
ADD   X3,  X1,  X2
ADDI  X4,  X1,  #15
NOP
ADD   X5,  X3,  X2
```

4.22: Page 375

(4.22.1)

In order to fit the table on the page, I am assigning each instruction to the variables A, B, C, D, E, and F respectively. Also, NOP will be N in the table.

CC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
A	IF	ID	EX	ME	WB																
B		IF	N	N	N	ID	EX	ME	WB												
C			IF	N	N	N	N	ID	EX	ME	WB										
D				IF	N	N	N	N	N	N	N	ID	EX	ME	WB						
E					IF	N	N	N	N	N	N	N	N	ID	EX	ME	WB				
F						IF	N	N	N	N	N	N	N	N	N	N	N	ID	EX	ME	WB

(4.22.2)

It is possible to rearrange the code in order to reduce the number of stalls by putting putting second line after the third line.

4.25: Page 376

(4.25.1)

CC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
LDUR	IF	ID	EX	ME	WB											
LDUR		IF	ID	EX	ME	WB										
ADD			IF	Stall	ID	EX	ME	WB								
SUBI					IF	ID	EX	ME	WB							
CBNZ						IF	ID	EX	ME	WB						
LDUR							IF	ID	EX	ME	WB					
LDUR								IF	ID	EX	ME	WB				
ADD									IF	Stall	ID	EX	ME	WB		
SUBI											IF	ID	EX	ME	WB	
CBNZ												IF	ID	EX	ME	WB

4.27: Page 378

(4.27.1)

```
ADD   X5,  X2,  X1
NOP
NOP
LDUR  X3,  [X5,  #4]
LDUR  X2,  [X2,  #0]
NOP
ORR   X3,  X5,  X3
NOP
NOP
STUR  X3,  [X5,  #0]
```

(4.27.2)

Even after attempting to rearrage the above code, the amount of NOP instructions remain the same. A reason for this, is because the instructions are dependent on the order in which they execute, therefore, one cannot rearrange the code.

(4.27.3)

If the processor has forwarding but with no hazard detection, when the code is being run, with forwarding you need the hazard system to detect problems, and without it, it won't be able to run because errors will run rampant because of hazards.

4.29: Page 379

(4.29.1)

With the provided branch outcomes of T, NT, T, T, NT we can assess the following for the case where it is always-taken and always-not-taken:

In the case where the predictors become T, T, T, T, T, there are only 3 out of the 5 that are correct, therefore, the accuracy is $\frac{3}{5}$ **or** 60%.

In the case where the predictors become NT, NT, NT, NT, NT, there are only 2 out of the 5 that are correct, therefore, the accuracy is $\frac{2}{5}$ **or** 40%.

(4.29.2)

1. The first branch will result in a T, moving from strong predict not taken to weak predict not taken. Predicted outcome is not taken. The predictor value is 0.
2. The second branch will result in a NT, moving from the weak predict not taken back to the strong predict not taken. Predicted outcome is not taken. The predictor value is 1.
3. The third branch will result in a T, moving from strong predict not taken to weak predict not taken. Predicted outcome is not taken. The predictor value is 0.
4. The forth branch will result in a T, moving from weak predict not taken to weak predict taken. Predicted outcome is not taken because it started out in the weak predict not taken. The predictor value is 0.

After looking at the 4 outcomes, only 1 of them were correct (second branch) so the accuracy is **25%**.