COMPENG 4DM4 Assignment 1 Report

Aaron Pinto
pintoa9

Raeed Hassan
hassam41

October 24, 2022

Exercise Part (A) - The 7-stage RISC Pipeline

(A1) There is a 2 clock-cycle stall on LOAD instructions when the instruction following a LOAD instruction uses the value being loaded. The timing diagram can be seen in Figure 1.

(A2) The number of branch-delay-slots following a BRANCH instruction will be 2 clock cycles in both cases considered:

1. If the value R0 being tested is available in a register at the start of the ID stage, then the "branch is resolved" in the ID stage - the ID stage will compare R0 with zero, and generate a "Take-Branch" signal which is sent back to the IF1 stage from the ID stage in clock-cycle 3. In this case, 2 branch-delay-slots will occur, as seen in Figure 2. The compiler can attempt to fill the "BRANCH+1" and "BRANCH+2" branch delay slots with useful instructions, otherwise fill with a NO-OP.

2. If the value R0 is being computed in the EX stage, by the previous instruction, and R0 needs to be forwarded to the ID stage, then we do not "stretch the clock" to allow R0 to be tested in the ID stage in the same clock cycle. The system will "stall" the BNEZ for 1 extra clock cycle, waiting for R0 to be computed and forwarded to the ID stage. In this case, 2 branch-delay-slots will occur, as seen in Figure 3. The compiler can attempt to fill the "BRANCH+1" and "BRANCH+2" branch delay slots with useful instructions, otherwise fill with a NO-OP.

	Clock Cycle														
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LW R0,0(R1)	IF1	IF2	ID	EX	MEM1	MEM2*	WB				*forward R0 (MEM2* to *EX) in cc6				
ADD R3,R0,R2		IF1	IF2	ID	stall	stall	*EX	MEM1	MEM2	WB					
LOAD+2			IF1	IF2	stall	stall	ID	EX	MEM1	MEM2	WB				
LOAD+3				IF1	stall	stall	IF2	ID	EX	MEM1	MEM2				
LOAD+4					stall	stall	IF1	IF2	ID	EX	MEM1	MEM2			

Figure 1: Timing Diagram for A1

	Clock Cycle											
Instruction	1	2	3	4	5	6	7	8	9	10	11	12
BNEZ R0, loop	IF1	IF2	ID*	EX	MEM1	MEM2	WB		*forward R0 (ID* to *IF1) in cc3			
BRANCH+1		IF1	IF2	ID	EX	MEM1	MEM2	WB				
BRANCH+2			IF1	IF2	ID	EX	MEM1	MEM2	WB			
BRANCH TARGET				*IF1	IF2	ID	EX	MEM1	MEM2	WB		

Figure 2: Timing Diagram for A2 Case 1

	Clock Cycle												
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13
previous instruction	IF1	IF2	ID	EX*	MEM1	MEM2	WB	*forward R0 (EX* to *ID) in cc4					
BNEZ R0, loop		IF1	IF2	stall	*ID**	EX	MEM1	MEM2	WB	**forward R0 (ID** to **IF) in cc5			
BRANCH+1			IF1	stall	IF2	ID	EX	MEM1	MEM2	WB			
BRANCH+2				stall	IF1	IF2	ID	EX	MEM1	MEM2	WB		
BRANCH TARGET						**IF1	IF2	ID	EX	MEM1	MEM2	WB	

Figure 3: Timing Diagram for A2 Case 2

Exercise Part (B) - Generate RISC Code for The Chacha20 Stream Cipher

(B1) The unoptimized RISC code for the first 3 lines of the QUARTER-ROUND operation is shown in Listing 1. The following assumptions are made for this RISC code:

- ASSUMPTION 1: assume that register R0 contains the address of the first word of the block of the initial key-stream
- ASSUMPTION 2: each word is 32 bits, so use LD and SD instead of LW and SD to load 32-bit words
- ASSUMPTION 3: words in the initial key-stream are stored consecutively in memory, address of second word is address of first world + 4 (bytes)
- ASSUMPTION 4: assume dual-ported memory that allows simultaneous read+read, read+write, write+write

Listing 1: Unoptimized RISC code for the first 3 lines of QUARTER-ROUND operation

```
1  % macros
2  #define Ra => R1; Rb => R2; Rd => R3     % define Ra, Rb, Rd
3
4  LD      Ra,(0)R0          % load a from memory
5  LD      Rb,(16)R0         % load b from memory
6  ADD     Ra,Ra,Rb          % a = a + b
7  SD      (0)R0,Ra          % store a in memory
8
9  LD      Ra,(0)R0          % load a from memory
10 LD      Rd,(48)R0         % load d from memory
11 XOR     Rd,Rd,Ra          % XOR(d,a)
12 SD      (48)R0,Rd         % store d in memory
13
14 LD      Rd,(48)R0         % load d from memory
15 ROT.L   Rd,Rd,#16         % ROTATE_LEFT(d, 16)
16 SD      (48)R0,Rd         % store d in memory
```

The timing diagram can be seen in Figure 4.

	Clock Cycle																											
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
LD Ra,(0)R0	IF1	IF2	ID	EX	MEM1	MEM2	WB																						
LD Rb,(16)R0		IF1	IF2	ID	EX	MEM1	MEM2*	WB	*forward Rb (MEM2* to *EX) in cc7																				
ADD Ra,Ra,Rb			IF1	IF2	ID	stall	stall	*EX	MEM1*	MEM2	WB	*forward Rd (MEM1* to *MEM1) in cc9																
SD (0)R0,Ra				IF1	IF2	stall	stall	ID	EX	*MEM1	MEM2*	WB	*forward (0)R0 (MEM2* to *MEM1) in cc11															
LD Ra,(0)R0					IF1	stall	stall	IF2	ID	EX	stall	*MEM1	MEM2	WB														
LD Rd,(48)R0						stall	stall	IF1	IF2	ID	stall	EX	MEM1	MEM2*	WB	*forward Rd (MEM2* to *EX) in cc14												
XOR Rd,Rd,Ra									IF1	IF2	stall	ID	stall	stall	*EX	MEM1*	MEM2	WB	*forward Rd (MEM1* to *MEM1) in cc16									
SD (48)R0,Rd										IF1	stall	IF2	stall	stall	ID	EX	*MEM1	MEM2*	WB	*forward (48)R0 (MEM2* to *MEM1) in cc18								
LD Rd,(48)R0											stall	IF1	stall	stall	IF2	ID	EX	stall	*MEM1	MEM2*	WB	*forward Rd (MEM2* to *EX) in cc20					
ROT.L Rd,Rd,#16													stall	stall	IF1	IF2	ID	stall	stall	stall	*EX	MEM1*	MEM2	WB	*forward Rd (MEM1* to *MEM1) in cc22		
SD (48)R0,Rd																IF1	IF2	stall	stall	stall	ID	EX	*MEM1	MEM2	WB			

Figure 4: Timing Diagram for B1

(B2) The unoptimized RISC code for one QUARTER-ROUND operation (along with the associated stall cycles per instruction) is shown in Table 1. We make the same assumptions as we did in B1. The following macros are defined:

Listing 2: B2 Macros

```
#define Ra => R1; Rb => R2; ; Rc => R3; Rd => R4          % define Ra, Rb
    , Rc, Rd
```

Table 1: Unoptimized RISC code for QUARTER-ROUND operation

Instruction	Number of stall cycles	Comment
LD Ra,(0)R0	0	load a from memory
LD Rb,(16)R0	0	load b from memory
ADD Ra,Ra,Rb	2	a = a + b
SD (0)R0,Ra	2	store a in memory
LD Ra,(0)R0	3	load a from memory
LD Rd,(48)R0	3	load d from memory
XOR Rd,Rd,Ra	3	XOR(d,a)
SD (48)R0,Rd	3	store d in memory
LD Rd,(48)R0	4	load d from memory
ROT.L Rd,Rd,#16	5	ROTATE_LEFT(d, 16)
SD (48)R0,Rd	3	store d in memory
LD Rc,(32)R0	3	load c from memory
LD Rd,(48)R0	3	load d from memory
ADD Rc,Rc,Rd	2	c = c + d
SD (32)R0,Rc	2	store c in memory
LD Rb,(16)R0	3	load b from memory
LD Rc,(32)R0	3	load c from memory
XOR Rb,Rb,Rc	3	XOR(b,c)
SD (16)R0,Rb	3	store b in memory
LD Rb,(16)R0	4	load b from memory
ROT.L Rb,Rb,#12	5	ROTATE_LEFT(b, 12)
SD (16)R0,Rb	3	store b in memory
LD Ra,(0)R0	3	load a from memory
LD Rb,(16)R0	3	load b from memory
ADD Ra,Ra,Rb	2	a = a + b
SD (0)R0,Ra	2	store a in memory
LD Ra,(0)R0	3	load a from memory
LD Rd,(48)R0	3	load d from memory
XOR Rd,Rd,Ra	3	XOR(d,a)
SD (48)R0,Rd	3	store d in memory
LD Rd,(48)R0	4	load d from memory
ROT.L Rd,Rd,#8	5	ROTATE_LEFT(d, 8)
		Continued on next page

Table 1 – continued from previous page

Instruction	Number of stall cycles	Comment
SD (48)R0,Rd	3	store d in memory
LD Rc,(32)R0	3	load c from memory
LD Rd,(48)R0	3	load d from memory
ADD Rc,Rc,Rd	2	c = c + d
SD (32)R0,Rc	2	store c in memory
LD Rb,(16)R0	3	load b from memory
LD Rc,(32)R0	3	load c from memory
XOR Rb,Rb,Rc	3	XOR(b,c)
SD (16)R0,Rb	3	store b in memory
LD Rb,(16)R0	4	load b from memory
ROT.L Rb,Rb,#7	5	ROTATE_LEFT(b, 7)
SD (16)R0,Rb	3	store b in memory

(B3) The unoptimized QUARTER-ROUND in B2 will take 82 clock-cycles to execute. We extended the the timing diagram for B1 (Figure 4) till after the next set of LOAD instructions, and discovered that the timing diagram repeats starting on the first arithmetic operation of the next column (as seen in Figure 5). This allowed us to extend the timing diagram to cover the entire odd ROUND operation, and we determined from the timing diagram that this operation would take 82 clock-cycles to execute.

	Clock Cycle																	
Instruction	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
ROT.L Rd,Rd,#16	stall	stall	IF1	IF2	ID	stall	stall	stall	EX	MEM1	MEM2	WB						
SD (48)R0,Rd				IF1	IF2	stall	stall	stall	ID	EX	MEM1	MEM2	WB					
LD Rc,(32)R0					IF1	stall	stall	stall	IF2	ID	EX	MEM1	MEM2	WB				
LD Rd,(48)R0						stall	stall	stall	IF1	IF2	ID	EX	MEM1	MEM2	WB			
ADD Rc,Rc,Rd										IF1	IF2	ID	stall	stall	EX	MEM1	MEM2	WB

Figure 5: Partial Timing Diagram for B2

(B4) When we optimize the QUARTER-ROUND code in B2, we are able to reduce the number of clock cycles required to execute the QUARTER-ROUND operation from 82 clock-cycles to just 26. The compressed timing diagram and the optimized QUARTER-ROUND code can be seen in Table 2. We make the same assumptions as we did in B1. The following macros are defined:

Listing 3: B4 Macros

```
#define Ra => R1; Rb => R2; ; Rc => R3; Rd => R4          % define Ra, Rb
    , Rc, Rd
```

(B5) The optimized RISC code for one ROUND which operates on the 4 COLUMNs of a block is simply the optimized QUARTER-ROUND operation from B4 repeated four times, but with different memory addresses used for each QUARTER-ROUND operation. The code for the ROUND operation is described in Table 3. We do not need to modify the code for

the QUARTER-ROUND operation as there is no stalling between the columns even if the code is repeated as-is. We make the same assumptions as we did in B1. In each iteration of the QUARTER-ROUND operation, the only changes are the memory addresses used for loading registers Ra, Rb, Rc, and Rd at the start of the QUARTER-ROUND operation. The following macros are defined:

Listing 4: B5 Macros

```
#define Ra => R1; Rb => R2; ; Rc => R3; Rd => R4          % define Ra, Rb
    , Rc, Rd
```

From the table, we can see that it would simply take 4 × the number of clock-cycles per QUARTER-ROUND operation or $4 \times 26 = 104$ clock-cycles.

Instruction	F1,F2	D	EX	M1,M2	WB	Comments
LD Ra,(0)R0	1,2	3	4	5,6	7	
LD Rb,(16)R0	2,3	4	5	6,7*	8	* forward Rb (M2 to EX) end of cc7
LD Rd,(48)R0	3,4	5	6	7,8*	9	* forward Rd (M2 to EX) end of cc8
LD Rc,(36)R0	4,5	6	7	8,9	10	
ADD Ra,Ra,Rb	5,6	7	*8**	9,10	11	** forward Ra (EX to EX) end of cc8
XOR Rd,Rd,Ra	6,7	8	*,**9***	10,11	12	*** forward Rd (EX to EX) end of cc9
ROT.L Rd,Rd,#16	7,8	9	***10*	11,12	13	* forward Rd (EX to EX) end of cc10
ADD Rc,Rc,Rd	8,9	10	*11**	12,13	14	** forward Rc (EX to EX) end of cc11
XOR Rb,Rb,Rc	9,10	11	**12*	13,14	15	* forward Rb (EX to EX) end of cc12
ROT.L Rb,Rb,#12	10,11	12	*13**	14,15	16	** forward Rb (EX to EX) end of cc13
ADD Ra,Ra,Rb	11,12	13	**14*	15,16	17	* forward Ra (EX to EX) end of cc14
XOR Rd,Rd,Ra	12,13	14	*15**	16,17	18	** forward Rd (EX to EX) end of cc15
ROT.L Rd,Rd,#8	13,14	15	**16*	17,18	19	* forward Rd (EX to EX) end of cc16
ADD Rc,Rc,Rd	15,16	16	*17**	18,19	20	** forward Rc (EX to EX) end of cc17
XOR Rb,Rb,Rc	17,18	17	**18*	19,20	21	* forward Rb (EX to EX) end of cc18
ROT.L Rb,Rb,#7	18,19	18	*19	20,21**	22	** forward Rb (M2 to ID) end of cc21
SD (0)R0,Ra	19,20	19	20	21,22	23	
SD (48)R0,Rd	20,21	20	21	22,23	24	
SD (32)R0,Rc	21,22	21	22	23,24	25	
SD (16)R0,Rb	22,23	**22	23	24,25	26	

Table 2: Optimized QUARTER-ROUND operation

Code	Clock Cycles	Comment
B4 QR((0)R0,(16)R0,(32)R0,(48)R0	26 clock cycles to execute 0 clock cycles for stalls	(0)R0, (16)R0, (32)R0, (48)R0 are the addresses for words 0, 4, 8, 12 (first column)
B4 QR((4)R0,(20)R0,(36)R0,(52)R0)	26 clock cycles to execute 0 clock cycles for stalls	(4)R0, (20)R0, (36)R0, (52)R0 are the addresses for words 1, 5, 9, 13 (second column)
B4 QR((8)R0,(24)R0,(40)R0,(56)R0)	26 clock cycles to execute 0 clock cycles for stalls	(8)R0, (24)R0, (40)R0, (56)R0 are the addresses for words 2, 6, 10, 14 (third column)
B4 QR((12)R0,(28)R0,(44)R0,(60)R0)	26 clock cycles to execute 0 clock cycles for stalls	(12)R0, (28)R0, (44)R0, (60)R0 are the addresses for words 3, 7, 11, 15 (fourth column)

Table 3: ROUND operation

(B6) The optimized RISC code and the outer and inner loop of Chacha20 can be seen in Table 4. For the DOUBLE-ROUND operation in the inner loop, the first ROUND operation

uses the unmodified code from B5, while the second ROUND operation updates the addresses for the diagonals. We also exclude the last STORE instruction from our diagonal ROUND operation, so we can place it after the BNEZ instruction to use one of our branch-delay-slots (there are 2 branch-delay-slots after each branch as shown in A2). We also fill our remaining branch-delay-slots with useful instructions which allows to have a Chacha20 operation with no stalling. The following assumptions are made for this RISC code:

- ASSUMPTION 1: assume that register R0 contains the address of the first word of the block of the initial key-stream
- ASSUMPTION 2: each word is 32 bits, so use LD and SD instead of LW and SD to load 32-bit words
- ASSUMPTION 3: words in the initial key-stream are stored consecutively in memory, address of second word is address of first world + 4 (bytes)
- ASSUMPTION 4: assume dual-ported memory that allows simultaneous read+read, read+write, write+write

The following macros are defined:

Listing 5: B6 Macros

```
#define Ra => R1; Rb => R2; ; Rc => R3; Rd => R4          % define Ra, Rb
    , Rc, Rd
#define Rlo => R5; Rli => R6    % define loop counters for outer and
    inner loops
```

Table 4: Outer and Inner Loops of Chacha20

Code	Number of stall cycles	Comment
LW.I Rlo,#1023	0	initialize loop counter for outer loop for 1024 blocks
outer_loop: LW.I Rli,#9	0	start of outer loop initialize loop counter for inner loop for 10 iterations of DOUBLE-ROUND
inner_loop: Insert RISC code from B5 for one round	0	start of inner loop odd round unmodified code from B5
Insert RISC code from B5 for one round with modified diagonal addressing excluding SD instruction	0	even round update B5 code to have addressing for diagonals (QR((0)R0,(20)R0,(40)R0,(60)R0), ...) exclude last store instruction to place in branch-delay-slot
BNEZ Rli,inner_loop	0	repeat inner_loop if inner loop counter not 0
SD (x)R0,Rb	0	use branch-delay-slot last store instruction from even ROUND
SUB.I Rli,#1	0	use branch-delay-slot decrement inner loop counter by 1
BNEZ Rlo,outer_loop	0	repeat outer loop if outer loop counter not 0
ADD.I R0,#64	0	use branch-delay-slot increment address to next block (4 * 16 bytes)
SUB.I Rlo,#1	0	use branch-delay-slot decrement outer loop counter by 1

(B7) For the optimized RISC code for a QUARTER-ROUND operation in B4, we had 20 instructions. Therefore, each ROUND operation contains 80 instructions. The inner loop in B6 contains 162 instructions (80 from first ROUND, 79 from second ROUND, 1 from branch, 2 from branch-delay-slots), and this iterates 10 times in every iteration of the outer loop for 1620 instructions. Each outer loop iteration contains 1624 instructions (1 from inner loop counter initialization, 1620 from inner loop, 1 from branch, 2 from branch-delay-slots), iterated 1024 times for a total of 1662976 instructions. Once we add the instruction for the outer loop counter initialization, and the 6 extra instructions for the last Chacha20 instruction to finish the WB stage of the pipeline, we are left with 1662983 instructions.

With a 2.5 GHz clock rate, we can process 2500000000 instructions per second. This means we can process 1662983 instructions in 0.665193 milliseconds, or 665.1932 microseconds.