PMPH - Assignment 2

Mxk62

September 25, 2016

1 CUDA Programming with Reduce and Segmented Scan

1.1 Task I.1

Exclusive scan and segmented exclusive scan has been implemented and tested. I've basically followed the same structure as the test given to us which tests scaninclusive. The tests uses a accumulator and sequentially adds, and compares the results. For segmented scan it resets after each segment.

2 Hardware Track Exercises

2.1 Task II.1a)

The registers is updated at the WB stage, and the next instructions reading / loading is in the ID stage, thus there is a need of three NOOPs when a instruction writes to a register which gets loaded from in the next instruction.

SEARCH:
LW R5, 0(R3) /I1 Load item
NOOP /R5 RAW HAZARD
NOOP /R5 RAW HAZARD
NOOP /R5 RAW HAZARD
SUB R6, R5, R2 /I2 compare with key
NOOP /R6 RAW HAZARD
NOOP /R6 RAW HAZARD
NOOP /R6 RAW HAZARD
BNEZ R6, NOMATCH /I3 check for match
ADDI R1, R1, #1 /I4 count matches
NOMATCH:
ADDI R3, R3, #4 /I5 next item
NOOP /R3 RAW HAZARD
NOOP /R3 RAW HAZARD
NOOP /R3 RAW HAZARD
BNE R4, R3, SEARCH /I6 continue until all items

2.2 Task II.1.b)

With a match:

Table 1: With a Match

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
LW R5, O(R3)	IF	ID	EX	ME	WB															
SUB R6, R5, R2		IF	ID	ID	ID	ID	EX	ME	WB											
BNEZ R6, NOMATCH			IF	IF	IF	IF	ID	ID	ID	ID	EX	ME	WB							
ADDI R1, R1, #1							IF	IF	IF	IF	ID	EX	ME	WB						
ADDI R3, R3,#4											IF	ID	EX	ME	WB					
BNE R4,R3,SEARCH												IF	ID	ID	ID	ID	EX	ME	WB	

- LW R5, O(R3) takes 1 Clock.
- SUB R6, R5, R2 : 4 clocks, because of RAW hazard on R5 from I1. The instruction is stalled 3 clocks and takes 1 to run.
- BNEZ R6 R6, NOMATCH: 4 clocks. RAW hazard on R6 from I2. The instruction is stalled 3 clocks and 1 to run.
- BNE R4, R3, SEARCH: 4 clocks, RAW Hazard on R3 from I5. It stalls 3 clocks and runs 1.

The two ADDI instruction requires no stalls as they do not use any registers dependent on the other instructions. The total runtime is: 1 + 4 + 4 + 1 + 1 + 4 = 15 clocks, when there is a match.

Without a match:

Table 2: Without a Match

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
LW R5, O(R3)	IF	ID	EX	ME	WB															
SUB R6, R5, R2		IF	ID	ID	ID	ID	EX	ME	WB											
BNEZ R6, NOMATCH			IF	IF	IF	IF	ID	ID	ID	ID	EX	ME	WB							
ADDI R1, R1, #1							IF	IF	IF	IF	ID	FLUSH	..							
ADDI R3, R3,#4											IF	FLUSH	..							
ADDI R3, R3,#4												IF	ID	EX	ME	WB				
BNE R4,R3,SEARCH													IF	ID	ID	ID	ID	EX	ME	WB

- LW R5, O(R3) takes 1 Clock.
- SUB R6, R5, R2 : 4 Clocks, because of RAW hazard on R5 from I1. The instruction is stalled 3 clocks cycles and takes 1 to run.
- BNEZ R6 R6, NOMATCH: 4 clocks. RAW hazard on R6 from I2. The instruction is stalled 3 clock cycles and 1 to run.
- BNE R4, R3, SEARCH: 4 clocks, RAW Hazard on R3 from I5. It stalls 3 clock cycles and runs 1.

The two flushed instructions each takes 1 clocks and we need to do I5 again, which sums in 3 clocks, which leaves us with: 1 + 4 + 4 + 4 + 3 = 16 clocks if there is not a match.

2.3 Task II.1.c

With register forwarding, which means the if a register is updated the write is forwarded in the ID stage, which means we can stall for one less clock in these cases.

With a match: As the forwarding allows us to decrease each instructions stalls by 1, we get the

Table 3: With a Match

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
LW R5, O(R3)	IF	ID	EX	ME	WB															
SUB R6, R5, R2		IF	ID	ID	ID	EX	ME	WB												
BNEZ R6, NOMATCH			IF	IF	IF	ID	ID	ID	EX	ME	WB									
ADDI R1, R1, #1						IF	IF	IF	ID	EX	ME	WB								
ADDI R3, R3,#4									IF	ID	EX	ME	WB							
BNE R4,R3,SEARCH										IF	ID	ID	ID	EX	ME	WB				

same equations as from II.1.b, but with 1 less clock for each stalled instruction. 12 clocks.

Without a match:

It is the same scenario for the case with a match, thus we get 13 clock cycles for without a match.

Table 4: Without a Match

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
LW R5, O(R3)	IF	ID	EX	ME	WB															
SUB R6, R5, R2		IF	ID	ID	ID	EX	ME	WB												
BNEZ R6, NOMATCH			IF	IF	IF	ID	ID	ID	EX	ME	WB									
ADDI R1, R1, #1						IF	IF	IF	ID	FLUSH	..									
ADDI R3, R3,#4									IF	FLUSH	..									
ADDI R3, R3,#4										IF	ID	EX	ME	WB						
BNE R4,R3,SEARCH											IF	ID	ID	ID	EX	ME	WB			

2.4 Task II.1.d

With full forwarding we are now able to get a updated register at the EX stage, which again means we can reduce the cycles for each stall by 1, giving us a 9 clocks with a match and 10 clocks without a match. This is because we can stall one less clock when stalling at the ID stage.

2.5 Task II.1.e

Unrolling the loop could increase the performance in terms of clocks, but may use more registers. It could be beneficial but it might serve other consequences such as having specific branching which does not handle uneven numbers. As for delayed branches I am currently uncertain whether or not it benefits, as it may just require us to stall more often depending on implementation.

2.6 Task II.2a

The loop might not be complete but the gist of it should be there:
ADDI R1, R0 1024

```
//stride
ADDI R2, R0 1

loop:
//loads the data with stride
LV V1 0(R3) R2
LV V2 0(R4) R2

//gets the new place to store
ADDI R3 R3
LV V3 0(R3) R2
ADDI R4 R4 512
LV V4 O(R4) R2

//Increments addresses
ADDI R3 R3 512
ADDI R4 R4 512

//dot product
MULV V5 V1 V2
MULV V6 V1 V2

//store it
SV V5 0(R5)
SV V6 0(R3)

//Update counter
SUBBI R1 R1 128
BNEZ R1 loop
```

2.7 Task II.2.b

We have the formula:

$$T_execution = LV + ADDI + MULV + SV + 512$$

We have that LV and SV each cost 30 cycles, ADDI 5 cycles and MULV 10 cycles and get:

$$30 + 5 + 10 + 30 + 512 = 587$$

2.8 Task II.2.c

I have not yet done this task.