APEX- PART 2- REPORT

Team Members: Brenda M Taveras Rodriguez, Gissella M Bejarano, Mehrnoosh Shakarami

---

First, we tried to divide each part to read about and extract questions about logic of that parts. We shared our questions in a google shared document. Then we implement different parts based on the following division:

- Fetch 1,2, Decode 1: Brenda
- Decode 2: Mehrnoosh
- Execute and Write-Back: Gissella
- Most of Debugging: Brenda, and a little by Gissella and Mehrnoosh
- Documentation: Mehrnoosh and Gissella
- Creating own test case and its step by step results (appendix 1) of all structures: Mehrnoosh

❖ **Notes:**

For now handle register X as a normal register regs[8]

When the Initialize method is called from the command line, the list of instructions is not re-grabbed from the text file

The simulation starts with the WB (where we update ARF from ROB.head) and continues the rest of the pipeline in order (Fetch 1, Fetch 2, Decode 1, Decode 2, Execute). This is to guarantee that an instruction that finishes in EX stage is not written to ARF in the same cycle.

Also we update the availability of the sources of IQ entries every time there’s a result to forward.

❖ **Structures used:**

Architectural registers: array of 8 entries

Physical registers (every array of 16 entries):

- array of values
- array of validity
- array of allocated

Rename table: array of 8 entries containing last physical register that renames each AR

Rename array: array of 16 elements indicating if that physical register is the last one in renaming some AR (used for deallocation).

LSQ: queue of maximum capacity of 4 entries

IQ: queue of maximum capacity of 8 entries

ROB: array of 17 elements to control the logic of head = tail

Prediction Backup vector: queue containing the branch instructions and the structures needed to recover tatus before branch in case of flushing.

❖ Logic

1- Fetch stage:

- F1: Fetch new instruction if :
  - Nothing is stall in F2
  - PC is among the permitted range
  - Increment PC address
- F2: Receive instruction from F1 if nothing is stall in D1

2- Decode:

Decode1:

- **HALT:** If instruction=”Halt”, then stop fetching. Else:
- **ROB:** See if there is free slot in ROB (check the status). We compare head and tail through % operation and one more slot in ROB).
- **PR:** Review all physical registers and de-allocate the ones that are valid and whose architectural register has been renamed. For all instructions but Store: See if there is a free physical register (physical register has been renamed and it is valid). The first physical register in the array is taken.
- **IQ:** For all instructions but Load/Store: See if there is free IQ (take the first one).
- **LSQ:** Only for Load/Store: see if there is space in LSQ. LSQ is defined as a queue of the same IQ structure.
- If there were slots in PR, IQ, ROB and LSQ (store-load): Create the entries in IQ, LSQ and ROB.

- **PR:** Allocate physical register for destination and set it as invalid. Get the indexes of physical registers that correspond to the source architectural registers.
- **RT:** Free physical register is recorded as new instance for architectural register (edit rename table). De-allocate physical register old instances.
- Set the **counter** in IQ and LSQ to 0 (explained in more detail in EX stage).
- **ZFLAG:** If instruction is arithmetical or logical update global variable lastInstrucPrIx, which holds the index of the last physical register that will or has updated the ZFlag. This is for BZ and BNZ can get the last arithmetic/logic instruction.
- **BRANCH:** Determine prediction based on offset (negative: taken; positive: not taken). Insert in the vector of Prediction Backup class no matter if we predict taken or not taken, add the current rename table, allocation and validity array of physical registers. Update PC by adding the literal to it in case of taken.
- Update **ROB**'s "whereS" field.

Decode2:

- **IQ-Sources:** Initialize all IQ sources validity in true. Read the physical registers available values in validity array and set the valid bit of IQ whose sources are those physical registers.
- **IQ-Tag:** Just set the tag in decode2 in the IQ entry if the value of the physical register used as a source is not valid, because we should not overwrite the value in the Ex stage forwarding part.
- **IQ-Counter:** no evaluation is made because just D1, EX.
- **BZ /BNZ:** set their first tag source with the address of lastInstrucPrIx and the second source corresponds to the literal, in the case of the literal the index will be equal to -1.
- Update **ROB**'s "where" field.

3- Execute:

- **IQ selection:** Go through all the FUs and grab the oldest instruction from the IQ waiting for the same FU and whose counter is 1. When passing from IQ to FU we free the IQ entry (iqEntryFree to 1). Set the IQ.counter to 1 for all valid entries of IQ and LSQ that didn't have its counter in 1. This is to guarantee that (as the stages are simulated in order) a result, that wasn't forwarded but already finish its execution, won't be selected to be executed in the next cycle and remain in D2.
- **INT FU:** If there's a ready instruction, move it to INT FU and set InstructionToForward as from INT.
- **MUL FU:** If there's a ready instruction, move instruction to first stage of MULT FU. If there was an instruction at first stage, move it to the second stage of MUL FU and so on until fourth stage. If an instruction finishes (MulCounter=4) then InstructionToForward as from MULT
- **MEM FU:** Pop first element of LSQ to MEM FU (for a STORE if it is at the top of ROB). If it is the first stage calculate the address and access memory. If one instruction finished, then InstructionToForward as from MEM.
- **FW:** The priority in case the three FUs have values to forward is MEM, MULT, INT (using for that the InstructionToForward variable). Set the sources of IQ entries that were waiting for the result as ready. Set IQ.counter as 1 (eligible for next cycle) of those entries just if the rest of the sources are valid too. The same for LSQ.
- **ROB:** Write all available results to ROB/Physical Registers and forward InstructionToForward's result to Decode2/IQ/LSQ. Also, write to the ZFLag array, the appropriate value.

- **BRANCH:**
    - If type of the branch is BZ, the Zflag was different from 0 and the prediction was taken (literal <0), flush all the instructions after the branch, and restore rename table, physical registers validity and allocation as they were before fetching the Branch AND update PC to instructionLine + 1
    - If type of the branch is BZ, the Zflag was equal to 0 and the prediction was not taken (literal >= 0), flush all the instructions after the branch AND restore rename table, physical registers validity and allocation as they were before fetching the Branch AND update PC to instructionLine + literal (the target address).
    - If type of the branch is BNZ, the Zflag was equal to 0 and the prediction was taken (literal <0), flush all the instructions after the branch AND restore rename table, physical registers validity and allocation as they were before fetching the Branch AND update PC to instructionLine + 1.
    - If type of the branch is BNZ, the Zflag was different from 0 and the prediction was not taken (literal >= 0), flush all the instructions after the branch, and restore rename table, physical registers validity and allocation as they were before fetching the Branch AND update PC to instructionLine + literal (the target address).
    - If they type of branch is JUMP or BAL then
      Flush all instructions after branch index (in ROB) to the tail of ROB
      Restore rename table, Physical register allocation, Physical register validation (only if for physical registers of instructions squashed from ROB)
    - FLUSH: if we have to flush the pipeline we have to clear the Branch vector. If we don't have to flush the pipeline just pop from the front (no matter if prediction correct or not).
    - Update ROB’s “where” field.

4- Write Back

- ROB head should be stored in ARF. Look for the physical registers valid bit and pass it to the ARF
- Pop the head of ROB.

Appendix 1: Test Case 1 (with results)

```
MOVC R7 0
MOVC R6 1
MOVC R4 2
MOVC R0 20000
MOVC R1 2
MOVC R2 4
SUB R2 R2 R6
BZ 3
ADD R7 R7 R4
JUMP R0 6
MUL R6 R1 R4
MUL R7 R1 R4
MOVC R5 6
HALT
```

Results

```
R7=0
R6=1
R4=2
R0=20000
R1=2
R2=4
R2=4-1=3
R7=0+2=2
Jump 20006
```

R2=3-1=2
R7=2+2=4
jump
R2=2-1=1
R7=4+2=6
jump
R2=1-1=0
R6=2*2=4
R7=4
r5=6

Test case 2 (step by step)

```
MOVC R0 100
MOVC R1 1
MOVC R2 3
MOVC R3 4
MOVC R4 5
MOVC R5 6
MOVC R6 7
MOVC R7 8
LOAD R3 R2 53
MUL R5 R3 R4
ADD R6 R3 R2
STORE R6 R2 50
MUL R2 R6 R3
BNZ -3
XOR R6 R2 R6
SUB R7 R4 R1
MUL R5 R6 R2
JUMP R7 20016
BZ 1
HALT
```

❖ Step by step result:

ROB:

CYCLE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
3	Movc0(H)	(T)															
4	Movc0(H)	Movc1	(T)														
5	Movc0(H)	Movc1	Movc2	(T)													
6	Movc0	Movc1(H)	Movc2	Movc3	(T)												
7	Movc0	Movc1	Movc2(H)	Movc3	Movc4	(T)											
8	Movc0	Movc1	Movc2	Movc3(H)	Movc4	Movc5	(T)										
9	Movc0	Movc1	Movc2	Movc3	Movc4(H)	Movc5	Movc6	(T)									
10	Movc0	Movc1	Movc2	Movc3	Movc4	Movc5(H)	Movc6	Movc7	(T)								
11	Movc0	Movc1	Movc2	Movc3	Movc4	Movc5	Movc6(H)	Movc7	Load	(T)							
12	Movc0	Movc1	Movc2	Movc3	Movc4	Movc5	Movc6	Movc7(H)	Load	MUL	(T)						
13	Movc0	Movc1	Movc2	Movc3	Movc4	Movc5	Movc6	Movc7	Load(H)	MUL	ADD	(T)					
14	Movc0	Movc1	Movc2	Movc3	Movc4	Movc5	Movc6	Movc7	Load(H)	MUL	ADD	(T)					
15	Movc0	Movc1	Movc2	Movc3	Movc4	Movc5	Movc6	Movc7	Load(H)	MUL	ADD	(T)					
16	Movc0	Movc1	Movc2	Movc3	Movc4	Movc5	Movc6	Movc7	Load	MUL(H)	ADD	STORE	(T)				
17	Movc0	Movc1	Movc2	Movc3	Movc4	Movc5	Movc6	Movc7	Load	MUL(H)	ADD	STORE	MUL2	(T)			
18	Movc0	Movc1	Movc2	Movc3	Movc4	Movc5	Movc6	Movc7	Load	MUL(H)	ADD	STORE	MUL2	BNZ	(T)		
19	Movc0	Movc1	Movc2	Movc3	Movc4	Movc5	Movc6	Movc7	Load	MUL(H)	ADD	STORE	MUL2	BNZ	(T)		
20	Movc0	Movc1	Movc2	Movc3	Movc4	Movc5	Movc6	Movc7	Load	MUL	ADD(H	STORE	MUL2	BNZ	(T)		

											)						
21	Movc0	Movc1	Movc2	Movc 3	Movc4	Movc5	Movc6	Movc7	Load	MUL	AD D	STO RE(H)	MUL 2	BNZ	A D D	(T)	
22	Movc0	Movc1	Movc2	Movc 3	Movc4	Movc5	Movc6	Movc7	Load	MUL	AD D	STO RE	MUL 2 (H)	BNZ	(T)		
23	Movc0	Movc1	Movc2	Movc 3	Movc4	Movc5	Movc6	Movc7	Load	MUL	AD D	STO RE	MUL 2 (H)	BNZ	(T)		
24	Movc0	Movc1	Movc2	Movc 3	Movc4	Movc5	Movc6	Movc7	Load	MUL	AD D	STO RE	MUL 2	BNZ (H)	X O R	(T)	
25	Movc0	Movc1	Movc2	Movc 3	Movc4	Movc5	Movc6	Movc7	Load	MUL	AD D	STO RE	MUL 2	BNZ	X O R(H)	S U B	(T)
26	Movc0(T)	Movc1	Movc2	Movc 3	Movc4	Movc5	Movc6	Movc7	Load	MUL	AD D	STO RE	MUL 2	BNZ	X O R(H)	S U B	MUL 3
27	JUMP	Movc1(T)	Movc2	Movc 3	Movc4	Movc5	Movc6	Movc7	Load	MUL	AD D	STO RE	MUL 2	BNZ	X O R	S U B(H)	MUL 3
28	JUMP	BZ	Movc2(T)	Movc 3	Movc4	Movc5	Movc6	Movc7	Load	MUL	AD D	STO RE	MUL 2	BNZ	X O R	S U B	MUL 3(H)
29	JUMP	BZ	HALT	Movc 3(T)	Movc4	Movc5	Movc6	Movc7	Load	MUL	AD D	STO RE	MUL 2	BNZ	X O R	S U B	MUL 3(H)
30	JUMP	BZ(T)	HALT	Movc 3	Movc4	Movc5	Movc6	Movc7	Load	MUL	AD D	STO RE	MUL 2	BNZ	X O R	S U B	MUL 3(H)
31	JUMP	BZ(T)	HALT	Movc 3	Movc4	Movc5	Movc6	Movc7	Load	MUL	AD D	STO RE	MUL 2	BNZ	X O R	S U B	MUL 3(H)
32	JUMP(T)	HALT	HALT(T)	Movc 3	Movc4	Movc5	Movc6	Movc7	Load	MUL	AD D	STO RE	MUL 2	BNZ	X O R	S U B	MUL 3
33	JUMP	HALT(H)	HALT(T)	Movc 3	Movc4	Movc5	Movc6	Movc7	Load	MUL	AD D	STO RE	MUL 2	BNZ	X O R	S U B	MUL 3

34	JUMP	HALT(H)	HALT(T)	Movc 3	Movc4	Movc5	Movc6	Movc7	Load	MUL	ADD	STORE	MUL 2	BNZ	XOR	SUB	MUL 3
35	JUMP	HALT	HALT(T) (H)	Movc 3	Movc4	Movc5	Movc6	Movc7	Load	MUL	ADD	STORE	MUL 2	BNZ	XOR	SUB	MUL 3

Rename Table:

Cycle	R0	R1	R2	R3	R4	R5	R6	R7	R8=X
1									
2									
3	0								
4	0	1							
5	0	1	2						
6	0	1	2	3					
7	0	1	2	3	4				
8	0	1	2	3	4	5			
9	0	1	2	3	4	5	6		
10	0	1	2	3	4	5	6	7	
11	0	1	2	~~3~~ 8	4	5	6	7	
12	0	1	2	~~3~~ 8	4	~~5~~ 3	6	7	
13	0	1	2	~~3~~ 8	4	~~5~~ 3	~~6~~ 9	7	
14	0	1	2	~~3~~ 8	4	~~5~~ 3	~~6~~ 9	7	
15	0	1	2	~~3~~ 8	4	~~5~~ 3	~~6~~ 9	7	
16	0	1	2	~~3~~ 8	4	~~5~~ 3	9	7	
17	0	1	5	~~3~~ 8	4	~~5~~ 3	9	7	
18	0	1	5	~~3~~ 8	4	~~5~~ 3	9	7	
19	0	1	5	~~3~~ 8	4	~~5~~ 3	9	7	
20	0	1	5	~~3~~ 8	4	~~5~~ 3	9	7	
21(UNDO 19,20)	0	1	5	8	4	3	9	7	
22	0	1	5	8	4	3	9	7	
23	0	1	5	8	4	3	9	7	
24	0	1	5	8	4	3	~~9~~ 10	7	

25	0	1	5	8	4	3	10	~~7~~ 9	
26	0	1	5	8	4	~~3~~ 11	10	9	
27									
28(UNDO 25,26)									
29 ... 35	DONE								

ALLOCATED:

	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P12	P13	P14
1	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
3	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
4	T	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F
5	T	T	T	F	F	F	F	F	F	F	F	F	F	F	F	F
6	T	T	T	T	F	F	F	F	F	F	F	F	F	F	F	F
7	T	T	T	T	T	F	F	F	F	F	F	F	F	F	F	F
8	T	T	T	T	T	T	F	F	F	F	F	F	F	F	F	F
9	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F	F
10	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F
11	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F
12	T	T	T	F->T	T	T	T	T	T	F	F	F	F	F	F	F
13	T	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F
14	T	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F
15	T	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F
16	T	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F
17	T	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F
18	T	T	T	T	T	T	T	T	T	T->F	F	F	F	F	F	F
19	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F
20	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F
21((UNDO 19..21)	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F

22	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F
23	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F
24	T	T	T	T	T	T	T	T	T	T->F	T	F	F	F	F	F
25	T	T	T	T	T	T	T	T	T	T	T	F	F	F	F	F
26	T	T	T	T->F	T	T	T	T -> F	T	T	T	T	F	F	F	F
27	T	T	T	F	T	T -> F	T	F	T	T	T	T	F	F	F	F
28 ... 35	DONE															

RENAMED:

	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15
11	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F	F
12	F	F	F	F	F	T	F	F	F	F	F	F	F	F	F	F
13	F	F	F	F	F	T	T	F	F	F	F	F	F	F	F	F
14	F	F	F	F	F	T	T	F	F	F	F	F	F	F	F	F
15	F	F	F	F	F	T	T	F	F	F	F	F	F	F	F	F
16	F	F	F	F	F	T	T	F	F	F	F	F	F	F	F	F
17	F	F	F	F	F	T	T	F	F	F	F	F	F	F	F	F
18	F	F	F	F	F	T	F	F	F	F	F	F	F	F	F	F
19	F	F	F	F	F	T	F	F	F	F	F	F	F	F	F	F
20	F	F	F	F	F	T	F	F	F	F	F	F	F	F	F	F
21	F	F	F	F	F	T	F	F	F	F	F	F	F	F	F	F
22	F	F	F	F	F	T	F	F	F	F	F	F	F	F	F	F
23	F	F	F	F	F	T	F	F	F	F	F	F	F	F	F	F
24	F	F	F	F	F	T	F	F	F	F->T	F	F	F	F	F	F
25	F	F	F	F	F	T	F	F-> T	F	T	F	F	F	F	F	F
26	F	F	F	F->T	F	T	F	F-> T	F	T	F	F	F	F	F	F
DONE																

IQ:

Cycle	0	1	2	3	4	5	6	7
3	Movc0							
4	Movc0	Movc1						
5	~~Movc0~~	Movc1	Movc2					
6	Movc3	~~Movc1~~	Movc2					
7	Movc3	Movc4	~~Movc2~~					
8	~~Movc3~~	Movc4	Movc5					
9	Movc6	~~Movc4~~	Movc5					
10	Movc6	Movc7	~~Movc5~~					
11	~~Movc6~~	Movc7						
12	MUL1	~~Movc7~~						
13	MUL1	ADD						
14	MUL1	ADD						
15	MUL1	ADD						
16	~~MUL1~~	ADD						
17	MUL2	~~ADD~~						
18	MUL2	BNZ						
19	MUL2	BNZ						
20	~~MUL2~~	~~BNZ~~						
21	ADD							
22	~~ADD~~							
23								
24	XOR							
25	XOR	SUB						
26	~~XOR~~	SUB						
27	JUMP	~~SUB~~						
28	JUMP	BZ						
29	~~JUMP~~	BZ						
30	~~JUMP~~	~~BZ~~						
31								
32	HALT							
33	HALT							

34	~~HALT~~							
35	DONE							

LSQ:

Cycle	0	1	2	3	4
11	Load				
12	LOAD				
13	~~Load~~				
14					
15					
16	STORE				
17	STORE				
18	~~STORE~~				
19..35					

Function Units:

Cycle	F1	F2	D1	D2	INT	MUL	MUL	MUL	MUL	MEM1	MEM2	MEM3
1	Movc0											
2	Movc1	Movc0										
3	Movc2	Movc1	Movc0									
4	Movc3	Movc2	Movc1	Movc0								

5	Movc 4	Movc 3	Movc 2	Movc1	Movc 0							
6	Movc 5	Movc 4	Movc 3	Movc2	Movc 1							
7	Movc 6	Movc 5	Movc 4	Movc3	Movc 2							
8	Movc 7	Movc 6	Movc 5	Movc4	Movc 3							
9	Load	Movc 7	Movc 6	Movc5	Movc 4							
10	MUL 1	Load	Movc 7	Movc6	Movc 5							
11	ADD	MUL 1	Load	Movc7	Movc 6							
12	STOR E	ADD	MUL 1	Load	Movc 7							
13	MUL 2	STOR E	ADD	MUL1(stall ed)						Load		
14	MUL 2	STOR E	ADD	MUL1(stall ed)							Load	
15	MUL 2	STOR E	ADD	MUL1(stall ed)								Load(F W)
16	BNZ	MUL 2	STOR E	ADD		MUL 1						
17	XOR	BNZ	MUL 2	STORE	ADD		MUL 1					
18	SUB	XOR	BNZ	MUL2(stall ed)				MUL 1		STOR E		
19	ADD	NOP	BNZ	MUL2(stall ed)					MUL 1		STOR E	
20	STOR E	ADD	NOP	BNZ		MUL 2						STORE
21	MUL	STOR E	ADD	NOP	BNZ		MUL 2					
22	XOR	NOP	NOP	NOP				MUL				

								2				
23	SUB	XOR	NOP	NOP					MUL 2			
24	MUL 3	SUB	XOR	NOP								
25	JUMP	MUL 3	SUB	XOR								
26	BZ	JUMP	MUL 3	SUB	XOR							
27	HALT	BZ	JUMP	MUL3	SUB							
28	NOP	HALT	BZ	JUMP		MUL 3						
29	NOP	NOP	HALT	BZ	JUMP		MUL 3					
30	HALT	NOP	NOP	NOP				MUL 3				
31	NOP	HALT	NOP	NOP					MUL 3			
32	NOP	NOP	HALT	NOP								
33	NOP	NOP	NOP	HALT								
34	NOP	NOP	NOP	NOP	HALT							
35	DONE											