Monish Devendran
B00817603

1. a). false

b). true

c). true

d). true

e). true

f). false

g). false

h). true

i) false

j) true

2. Load store Queue operates as circular FIFO:
By using a separate LSQ in the ROB we can achieve certain thing:-
1. By using separate LSQ we can allow more than one Load / store instruction into the queue by the tail pointer in LSQ indicating the most recent instance, therefore stall other operation at dispatch will be fast and other reg - to - reg instruction will be sent to respective Function units.
2. Each LSQ entry has it's own fields set up ~~at~~ Whenever a load/store instruction is renamed, it is placed in The ROB and LSQ. Hence by using LSQ seperately both Load / store instruction uses register to computer effective address then transfer data or data from address in memory into register, the effective address or most recent instance is written in LSQ therefore Whenever there is ~~de~~ memory dependence between the usage of seperate LSQ will help's us to find The memory dependence
3. use of Seperate LSQ maintain all-inflight memory instrucion in program order.

3.

a). In the variation 3 of both the physical and architectural register are implement in common register file. A physical register allocated for destination of an instruction immediately after the instruction commitment. So if there is renamer. we can use a locally checkpoint at CRF, ~~So when a instruction~~
In the given code example:-
The renaming hardware to efficiently deallocate the physical register of R5.
Instruction ADD defines R5, creating a mapping to a physical register P9, instruction MUL is last use of R5. However P9 cannot be freed until R5 is redefined in instruction MUL. In meantime SUB instruction can pass between the last use P9 (R5) and its deallocation.
We can have active lists to track all uncomitted instruction in ~~per thread~~ program order.

In the CRF has the destination register and it's corresponding physical register. ~~whe~~ For each instruction retirement it will check it previous physical register for the particular destination address if the current instruction is commited it will deallocate the previous associative Physical register assigned after commitement.

For example : When MUL instruction is about to commit it checks R5 previous physical register as ADD R5 is already commited it gets P9 then P9 can be deallocated when MUL commits.

b) ROB Format :-

~~S~~ ROB → Status

ROB → itype (r2r)

ROB → PC_value (address of dispatched instruction)

ROB → ar_address (address of destination)

ROB → excodes

ROB → p_reg_dest (physical register for the destination register)

~~ROB~~ Status will be 1 after the instruction is commited and if Status is 1 it will sent to rename table if destination address with previous physical registee address also be sent.

if (ROB. head == ~~Rename~~
t

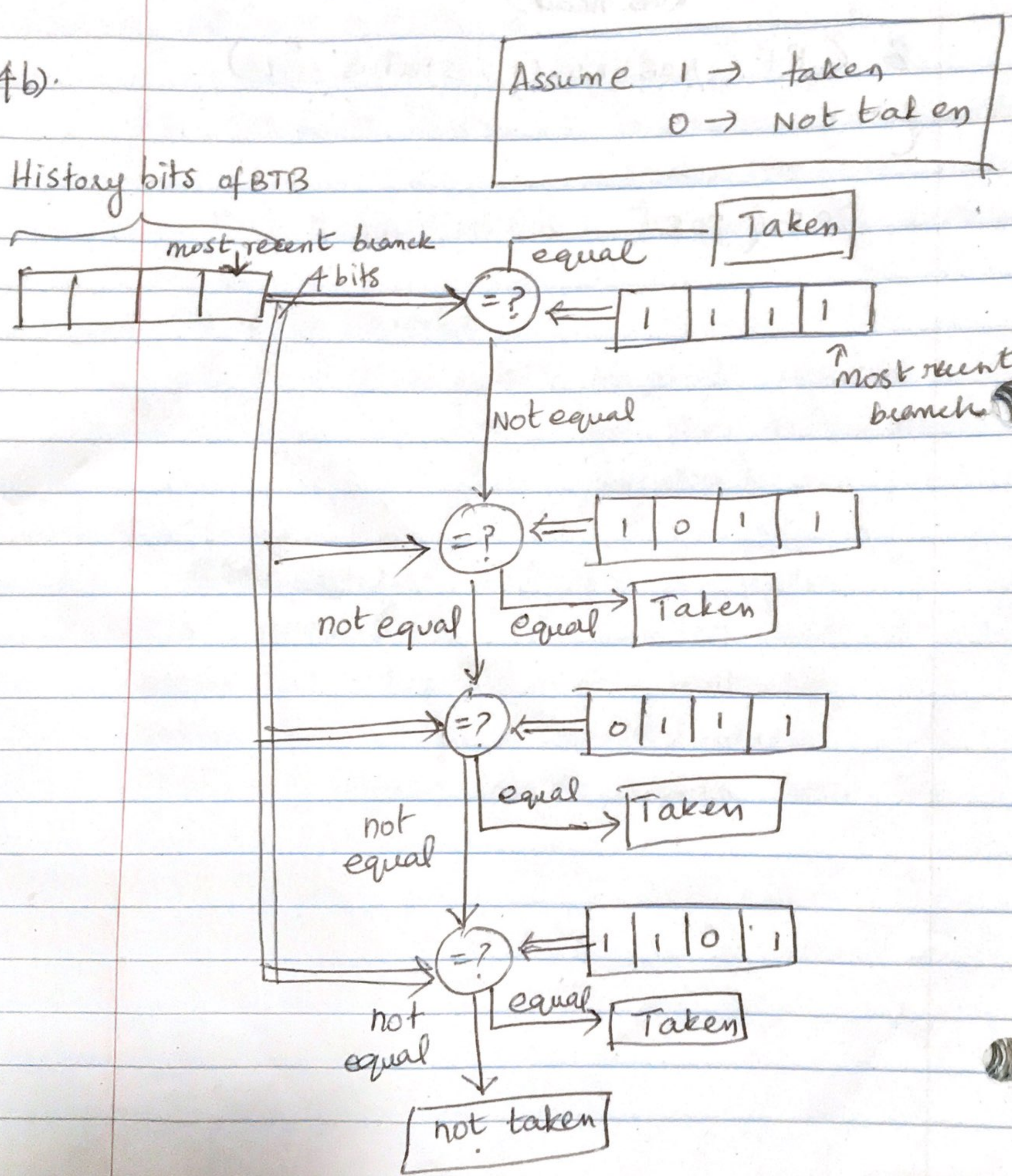
c). the result in ROB head is given to ARF that is pointed by retirement register alias table and ROB Slot id and src bit update and head is increment. then check the previous physical register mapped to that ar_address to deallocate it

```
if (ROB. head == ARF (dest_reg)
   && (ARF ( Status ==1) )

{
PRF [ROB . p-reg_dest] = 0

ARF [ar_address ] = Result from FU
ARF [src_bit] = 0

}
head ++
```

d). Yes there will be a ~~[illegible]~~ performance issue while commiting to check the physical register of the previous instruction that was committed before the current instruction

4. a). associative lookup tags;
history bits (4 bits);
effective address of target.

4b).


Assume 1 → taken
0 → Not taken
History bits of BTB
most recent branch
4 bits
=?
equal
Taken
1 1 1 1
most recent branch
Not equal
=?
1 0 1 1
equal
Taken
not equal
=?
0 1 1 1
equal
Taken
not equal
=?
1 1 0 1
equal
Taken
not equal
not taken