NAME: Monish Devendran

BU Number: B00817603

1.a. Various fields of ROB at the time of dispatch for load:

ROB[ROB.tail].status = invalid;
ROB[ROB.tail].itype = LOAD;
ROB[ROB.tail].PC_VALUE = 1024;
ROB[ROB.tail].ar_address = R2;
ROB[ROB.tail].excodes = left as it is (will be updated after instruction completion)
ROB[ROB.tail].result = left blank (will be updated after instruction completion)
ROB.tail ++;

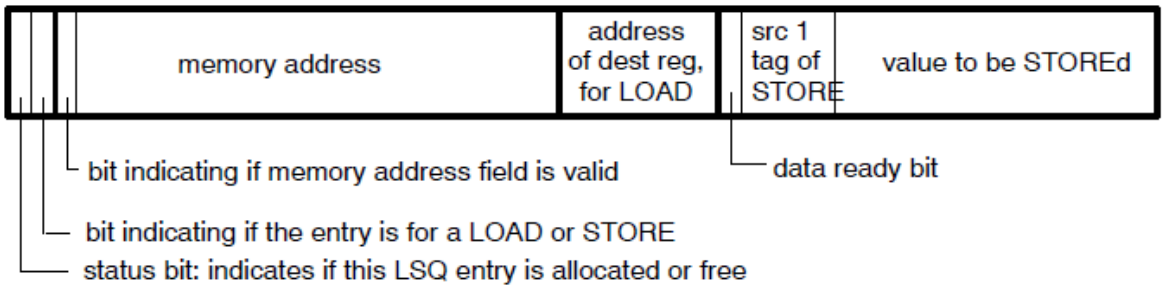
1.b. When load instruction is dispatched the entry at the rename table will be as follows:

RENAME_TABLE[R2].ar/slot_id = ROB.tail (i.e , 38)

RENAME_TABLE[R2].src_bit = 1 (because its not yet written to the ARF)

1.c.

Format of LSQ as follows:



1	LOAD	1	Once IQ entry for LOAD issues, the targeted memory address R5+20 is written.	R2	0	0	0

Make sure that memory address of LSQ are filled after the IQ entry where the LSQ index is been set.

1.d.

dest_slot_id = R2

ROB[dest_slot_id].result = 2470

ROB[dest_slot_id].excodes = FLAG_VALS

ROB[dest_slot_id].status = valid

These entry fields are set at the ROB during the instruction completion which just write the results and exception codes produced. The FLAG_VALS will be user handled based on the flag value it handles the exceptional codes for instruction handling.

1.e.

The value of 38 will be feeded to the ROB then to ARF and sets it corresponding entries and will fail the if statement as the **most recent entry at rename table will be 12** at the time of commitment. So the value the most recent value of 38 will be at ROB and at the rename table with sbit 1 and it will be copied to the 12 from **destination register R2 from rob** and it will maintain as the recent value and checks the if condition back again now it succeeds and writes the result to ARF and sets the sbit at rename table to 0.

1.f

When the load is committed the rob entry field will have the following entries:

Initially the rename table entry will look like at the time of dispatch:

Ar/slot_id	src bit
38 (R2)	1

R2 is the 38th slot in the ROB.

When the load instruction is commited the following changes are made as follows: When the instruction at the commit head is no longer busy and its not expecting it may be committed, i.e changes are made to ARF are made visible once commitment and then the reorder buffer entry is freed.

So basically in simple words : . Commit—update register with reorder result When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr from reorder buffer.

Ar/slot_id	src bit
38 (R2)	0

Now the R2 value can be read directly from the ARF (Architectural Register File) as the src bit is set to 0.

C code : ref. page 193 of notes

```
ARF[ROB[ROB.head].ar_address]] = ROB[head].result;
if ( ((Rename_Table[ROB[head].ar_address].ar/slot_id
== ROB.head)
& (Rename_Table[ROB[head].ar_address].src_bit ==1) )
then {
    Rename_Table[ROB[head].ar_address].ar/slot_id =
            ROB[head].ar_address;
    Rename_Table[ROB[head].ar_address].src_bit = 0
};
ROB.head++
```

Line 1 . is writing the result from ROB to the ARF(architectural register file) based on ROB head destination address which is R2.

Line 2 . If condition to make sure the address of destination slot id from rename table and the rob index value for head are equal i.e (38 == 38) AND the src_bit is 1 for the destination address R2 as it is not update to the ARF still it is at ROB.

Line 3. If the condition passes then set the Rename table slot id with the destination address from ROB which is R2. And then set the src_bit to 0 as it is written to Architectural register file.

Line 4. Increment the head index of the ROB to point to next.

1.g LSQ.head = 22

ROB.head = 38

2. a. Reasonable **format for ROB** entry for variation 2 will be as follows:

Status (indicating if the result/address is valid) -> ***status***

Instruction type (reg-to-reg , load , store, etc..) -> ***itype***

Instruction address -> ***PC_value***

Exception codes -> ***excodes***

Physical register (will be mapped to anyone of the physical register which will store the address of the desination register) -> ***p_reg***

2.b.

Instruction to be dispatched : LOAD <dest> <src1> <literal>

Issue Queue (IQ):

Status = valid or 1

A bit to indicate is the entry is **LOAD** instruction.

Literal operand field : **<literal>**

Src1 ready bit = 1

Src tag = **<src1>**

Src1 value = <src1 value>

LSQ index = will have the index value at which the memory operations should be calculated.

LOAD STORE Queue (LSQ):

Status bit = 1 or valid

Type = LOAD

Valid_bit =1

Memory_address = <src1+literal> (this will be returned from the IQ at the time of dispatch)

Dest register for load = <dest>

Reorder Buffer (ROB):

status = invalid

itype = LOAD

PC_value = based on instruction memory address

p_reg = address of the destination register to be mapped at the PRF.

Exception code = will be updated after the instruction completion.

2.c **Format of the rename table** in the variation 2 will be as follows:

The rename table is replaced as PRF (physical register file). Hence results will be directly updated to the PRF based on the ROB directly from the respective functional units.

Status_bit = indicating the PRF is allocated or not

Valid_bit = indicating the result to updated is valid or not by seeing the ROB.

ARF_dest_address = this field is used to map the result to the correct ARF destination register address.

Result = value to be updated to the destination register.

2.d When a load is committed, the memory is calculated with respect to LOAD FU and then based p_reg assigned at ROB, the result is updated on the PRF (physical register file) in which the result is sent to the correct destination address in ARF based on the ARF_dest_address field at the PRF, then the value in LSQ and ROB are updated for the next PC instruction.

3. a. BNZ 01 # -40

When it is at the ending of the loop it might have predicted to that the branch is taken which is wrong and hence it should squash the branched instruction value and load the correct instruction once the branch not taken.

So basically, it miss predicts to be branch taken and it loads the targeted branch instruction in the fetch and decode stage where the bnz is at ex stage, so it should squash the targeted instruction and should load the branch not taken instructions in the pipeline. So this false under, 01 : squash if the branch is not take (this implies that the **static prediction is that branch is taken most of the time**)

3. b.

```
        MOVC R6, #0                     /* R6 ← 0 */
        MOVC R7, #0
        MOVC R8, #400                   /* number of iterations in R8 */
1028:   LOAD R4, R1, #0          /* loop body starts here */
        LOAD R5, R2, #0
        SUB R10, R4, R5
        STORE R10, R1, #0
        ADD R10, R4, R5
        STORE R10, R2, #0
        ADDL R1, R1, #4
        ADDL R2, R2, #4
        ADDL R7, R7, #1
        CMP R7, R8                      /* compare contents of R7 and R8 and set CC flags */
1064:   BNZ # -40
1068:   STORE R1, R2, #0         /* first instruction after loop body */
        MUL R10, R12, R14
```

Transformed/Modified code:

MOVC R6, #0

MOVC R7, #0

MOVC R8, #400

LOAD R4, R1, #0

LOAD R5, R2, #0

SUB R10, R1, #0

STORE R10, R1, #0

ADD R10, R4, R5

STORE R10, R2, #0

ADDL R7, R7, #1

CMP R7, R8

BNZ 01, #-40

NOP # /*FILLER*/

ADDL R1, R1, #4

ADDL R2, R2, #4

STORE R1, R2, #0

MUL R10, R12, R14

The two instruction ADDL R1, R1, #4 and ADDL R2, R2, #4 can be moved to the delayed slot because these can be executed without the effects of a preceding instruction and also these two ADDL instructions that are not dependent on the result of the branch instruction.