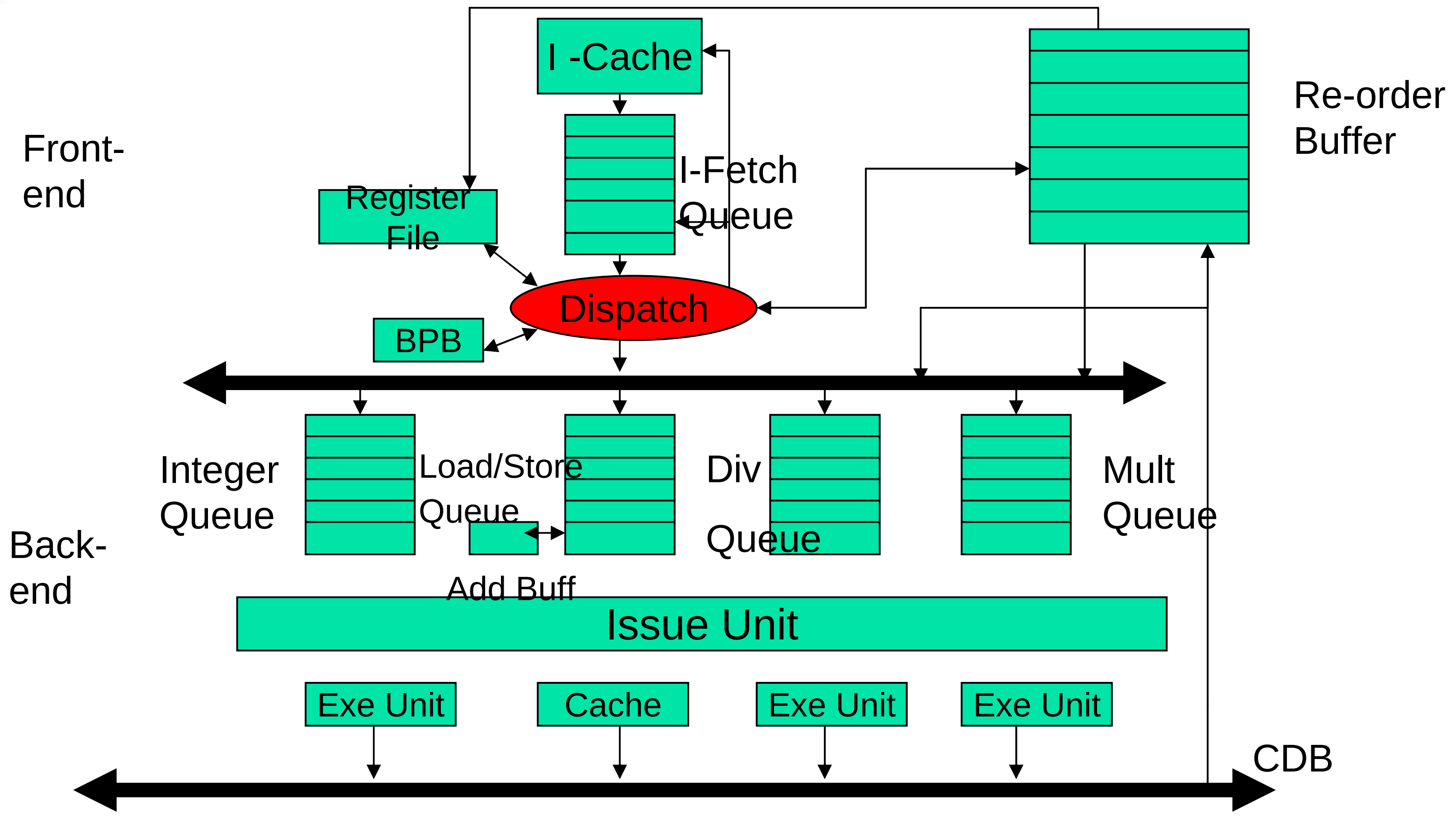
DISPATCH UNIT

Dispatch Overview



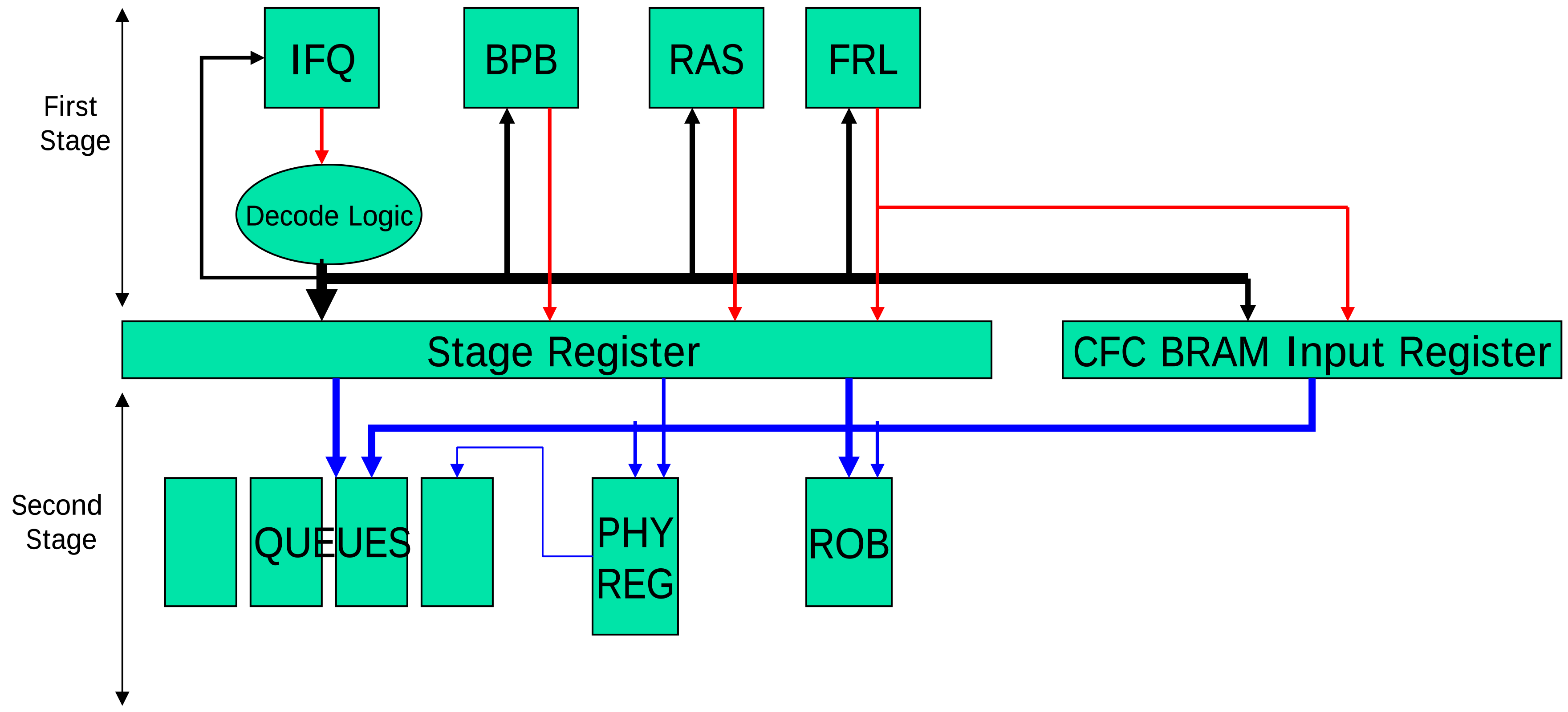
Dispatch Overview

- Get instruction from IFQ (one instruction at a time in process order).
- Decode the instruction (R-Type, Lw/Sw, Div, Mul, Jump,...etc).
- Rename source and destination registers.
  - The logical address of each source register, namely $rs and $rt are provided to CFC. CFC holds the table that stores the mapping from logical to physical registers , the Register Alias Table (RAT).
  - For register writing instructions, free register list (FRL) provides the address of the free physical register that will be mapped to the destination logical register. In addition, the old physical register which was mapped to the destination register is read from RAT in order to be re-claimed when the instruction commits.
- Allocate one ROB entry and one instruction issue queue entry if needed. For example, Jump instruction is executed in the dispatch and hence no need for ROB and issue queue allocation.
- Dispatch instruction to appropriate issue queue entry.

2-Stage Dispatch

- CFC mechanism requires a single committed RAT -which contains mapping of the latest committed instructions- and multiple incremental RATs (in our project: 8). Every time a branch or JR $31 instruction is dispatched a new incremental RAT is allocated and is called active RAT while all other are freezed. All new mappings are stored in the active RAT.
- Due to resource constraints, we implemented RATs as BRAMs within the CFC unit. BRAMs have input registers and hence it is not possible to rename the source and destination registers in a single clock. This requires the dispatch unit to be pipelined in 2 stages.

2-Stage Dispatch Unit



1st Stage Overview

- Receives input signals from all other units. All the decision making logic is in the first stage.
- Interacts with IFQ , RAS , BPB , FRL and CFC.
- Stalls instructions; instructions are only stalled in 1st dispatch stage. Conditions for stall are discussed in detail in slide (9).

Interaction with IFQ, RAS, BPB, FRL and CFC

- Fetch the instructions from IFQ in process order one at a time.
- Decode the instruction to identify its type and extract the logical addresses of source and destination registers.
- Access BPB to predict the direction of conditional branches. If branch is predicted Taken then we fetch from the target address of the branch, otherwise we continue fetching from the next PC value.
- Access return address stack (RAS) to get the jump address prediction of JR $31 instructions. Notice that this is only a prediction, it may fail and in that case we need to flush all younger instructions and start fetching from the correct address provided thru software RAS.

Interaction with IFQ, RAS, BPB, FRL and CFC

- Communicate with FRL to get an address of a free physical register that can be mapped to the destination logical register for register writing instruction. For example, LW and R-Type instructions write their results to $rt and $rd respectively. Hence, whenever a LW or an R-Type instruction is dispatched we should have at least one free physical register to which $rt or $rd will be mapped.

- Provide CFC unit with the logical addresses of source registers. In addition, for register writing instruction we need to provide both logical and physical register addresses of the destination.

Instructions Stall

- ROB is full, and the instruction requires an ROB entry.
- ROB has a single ROB entry and the instruction in the 2nd stage is valid (requires an ROB entry).
- The desired issue queue is full.
- The desired issue queue has a single entry and the instruction in the 2nd stage is of the same type and hence will be using that entry.
- IFQ is empty (no more instruction to fetch).
- FRL is empty (no free physical register available) and the instruction is a register writing instruction.
- CFC is full, all incremental RATs are used and the instruction is either a branch or Jr $31.
- Jr $rs instruction stalls the dispatch until the value of $rs is on CDB in order to calculate the jump address.

Instruction Stall- Special Case: Jr $rs instruction

- 1-bit flag register “Jr_stall” .
- 5-bit internal register for storing the RobTag of Jr $rs instruction “jr_rob_tag”:
  - Only used with Jr $rs.
  - Compared with RobTag on cdb.
  - If there is a match then Jr $rs has finished execution and jump address is ready on the Cdb. Hence , we can come out of stall.

2nd Stage Overview

- Interacts with CFC, Issue Queues , ROB and Physical Register File (PRF).
- Instructions don’t stall in this stage. They are ensured to be dispatched.
- Instructions will be flushed in case of Cdb_Flush.

Interaction with CFC, PRF, Issue Queues, and ROB

- CFC checks active, frozen and committed RAT to get the mappings of source registers. For register writing instructions, CFC stores the new mapping of the destination register in the active RAT and provides old mapping to re-claim previous physical register at commit.

- The physical register addresses of the source registers (provided by CFC) are fed to the physical register file (PRF) to check whether the values of the registers are ready or pending in the back end.

- An ROB entry is assigned and related instruction information are stored there.

- An entry in the desired issue queue is assigned and the related instruction information are stored there.

ROB Allocation

- An ROB entry is allocated for each instruction in the 2nd dispatch stage except in the following cases:
  - Stalling in the 1st dispatch stage causes the instruction in the 2nd stage to be invalid and hence no ROB entry is allocated.
  - Jmp Instruction finishes execution in the 1st dispatch stage and becomes an invalid instruction in 2nd dispatch stage, so it does not require an ROB entry.
  - Cdb Flush.
  - Special Case:
    - JR $rs instruction causes the design to stall until the address of the Jr $rs is computed. Although JR $rs does not require an ROB entry, we assign the RobTag determined by the ROB bottom pointer and keep snooping on the Cdb waiting for that RobTag.

Instructions Supported

- Add , Sub , and , or , slt
- Addi
- Load
- Store
- Mul
- Div
- Jump , Jal , Jr $31 , Jr $rs
- Beq , bne

JAL Instruction

- Jump address is given to IFQ and it is informed to fetch from the new address.
- PC + 4 is given to RAS to push on the stack.
- Integer queue is informed it’s a register writing instruction.
- PC + 4 is sent on Dis_BranchOtherAddr pins to be written in the new physical register assigned for $31.

JR $31 Instruction

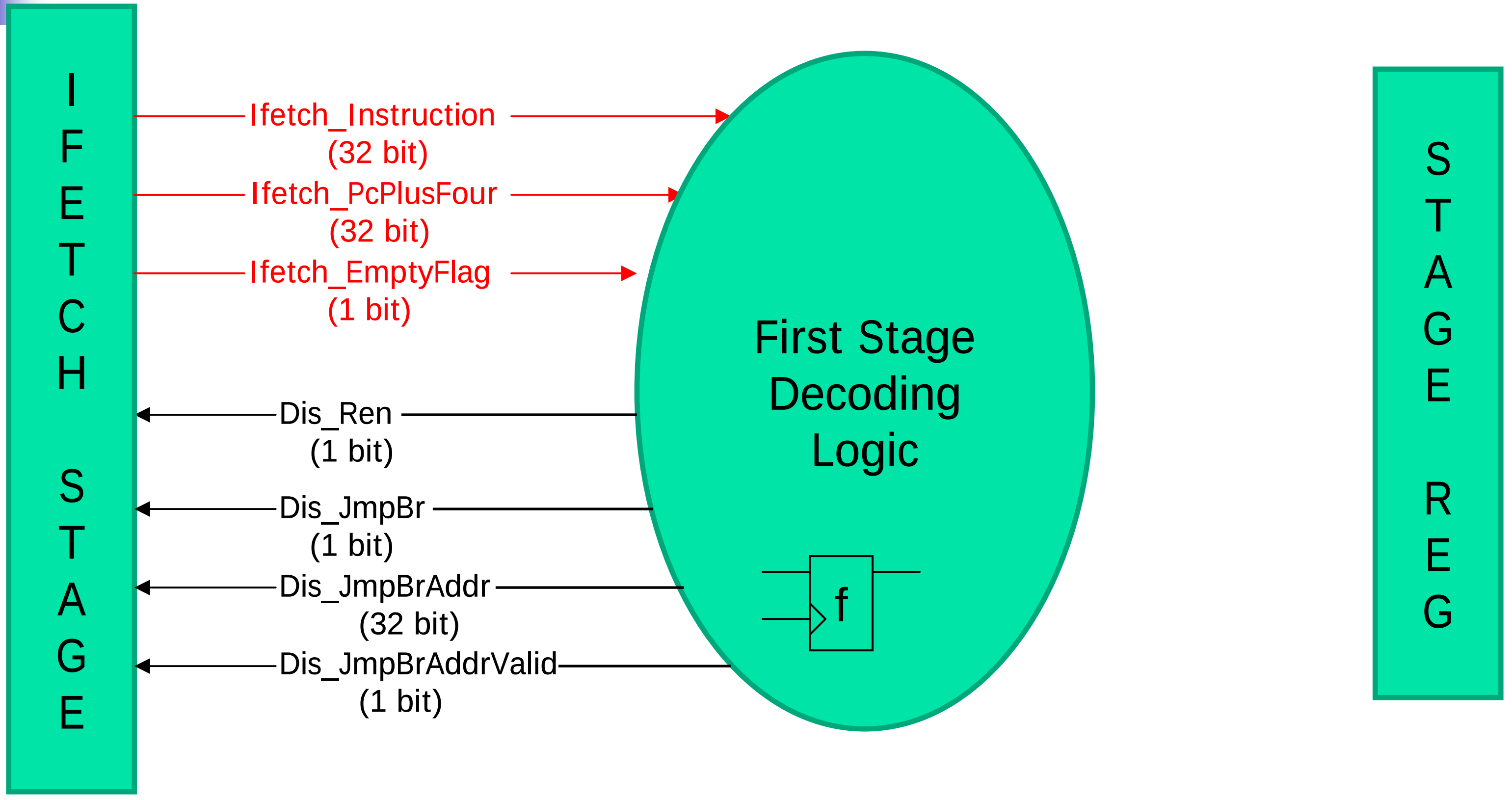
- RAS and CFC are informed it’s a JR $31 instruction.
- CFC creates a new checkpoint for the instruction.
- Ras_Addr is taken from RAS and given to IFQ to jump
- Ras_Addr can be invalid.
- Integer queue is informed it’s Jr $31 instruction and Ras_Addr is sent on Dis_BranchOtherAddr pins along with the physical register mapped to $31.
- ALU compares the Ras_Addr sent and physical register value and causes Cdb_Flush in case they are different

JR $Rs Instruction

- Integer Queue is informed it’s a JR rs instruction.
- Dispatch stores the RobTag in a 5 bit internal register and stalls.
- No valid entry is made in Rob as we don’t want to waste an Rob slot.
- Dispatch stalls till the instruction comes on Cdb with the value stored in the physical register mapped to $Rs. IFQ jumps to that address.
- Cdb_Flush prevails stall due to JR $rs instruction.

Interaction with other units..

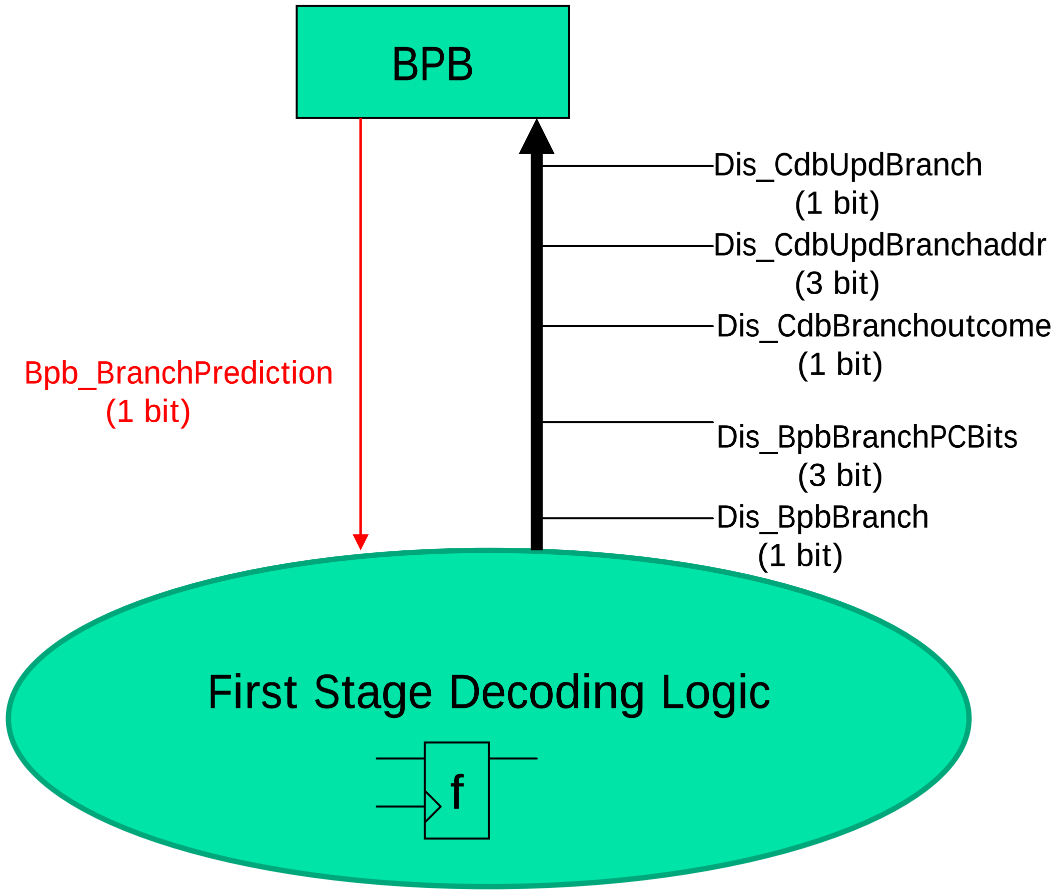
With Ifetch



IFETCH STAGE
Ifetch_Instruction (32 bit)
Ifetch_PcPlusFour (32 bit)
Ifetch_EmptyFlag (1 bit)
Dis_Ren (1 bit)
Dis_JmpBr (1 bit)
Dis_JmpBrAddr (32 bit)
Dis_JmpBrAddrValid (1 bit)
First Stage Decoding Logic
f
STAGE REG

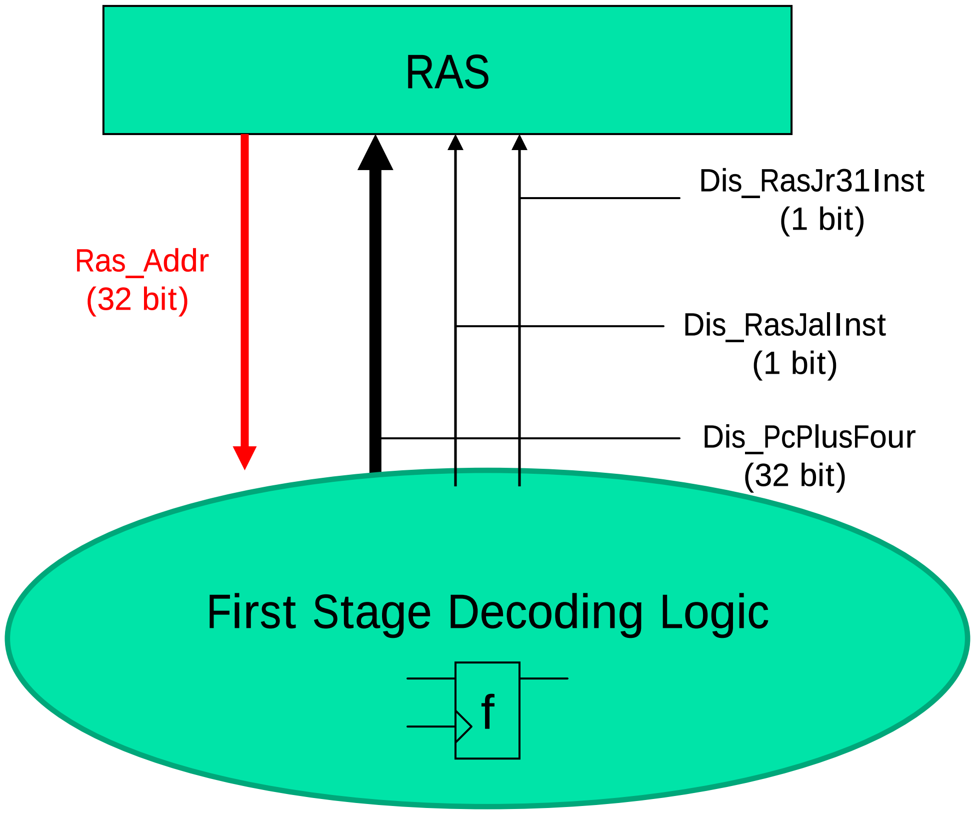
NOTE: "f" represents the flag register and 5-bit internal tag register of first stage for storing **stall** information.
Dis_JmpBrAddrValid signal is needed to distinguish between valid and invalid branch/jmp address on address pins. The address is valid for branch , jmp , jal and jr $31 instruction. And it is invalid for jr $rs instruction.

With BPB

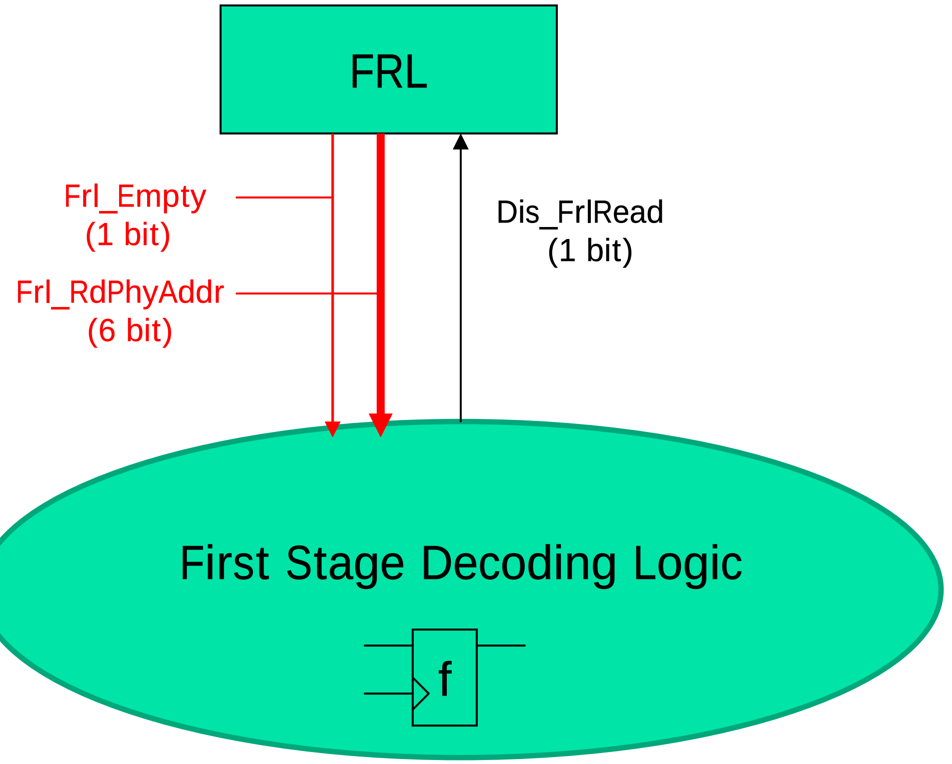

BPB
Bpb_BranchPrediction (1 bit)
Dis_CdbUpdBranch (1 bit)
Dis_CdbUpdBranchaddr (3 bit)
Dis_CdbBranchoutcome (1 bit)
Dis_BpbBranchPCBits (3 bit)
Dis_BpbBranch (1 bit)
First Stage Decoding Logic
f

NOTE: BPB gets updated from CDB rather from ROB top

With RAS

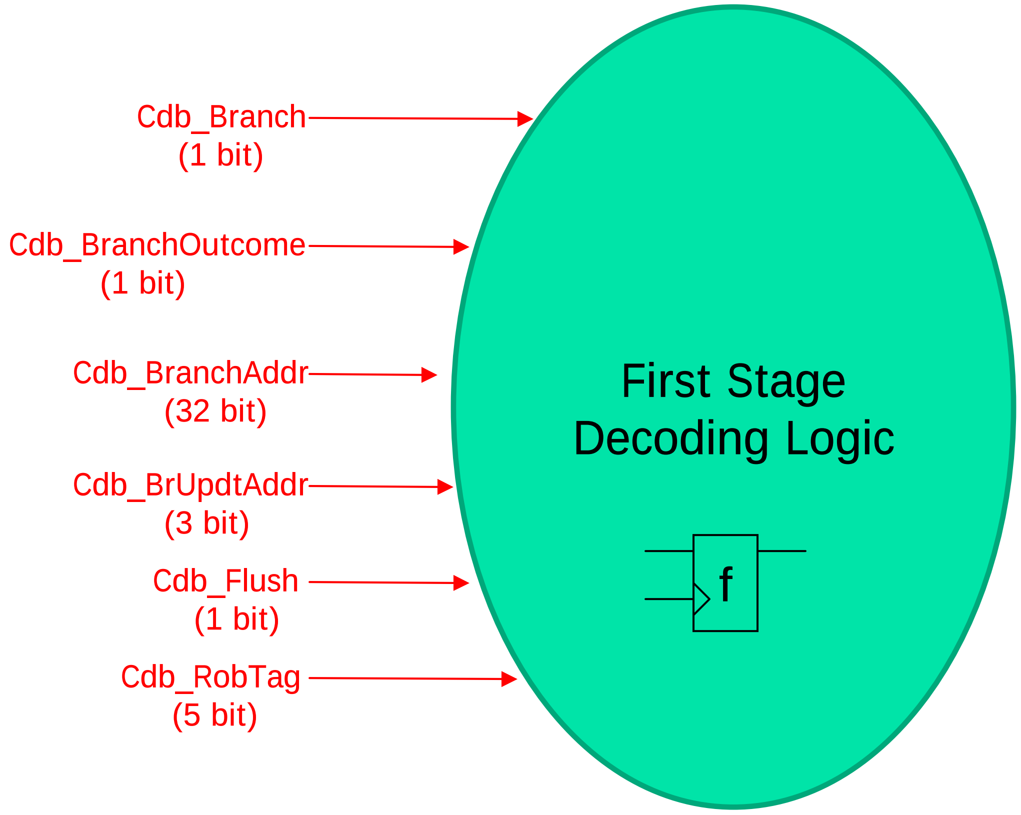

RAS
Ras_Addr
(32 bit)
Dis_RasJr31Inst
(1 bit)
Dis_RasJalInst
(1 bit)
Dis_PcPlusFour
(32 bit)
First Stage Decoding Logic
f

With Frl



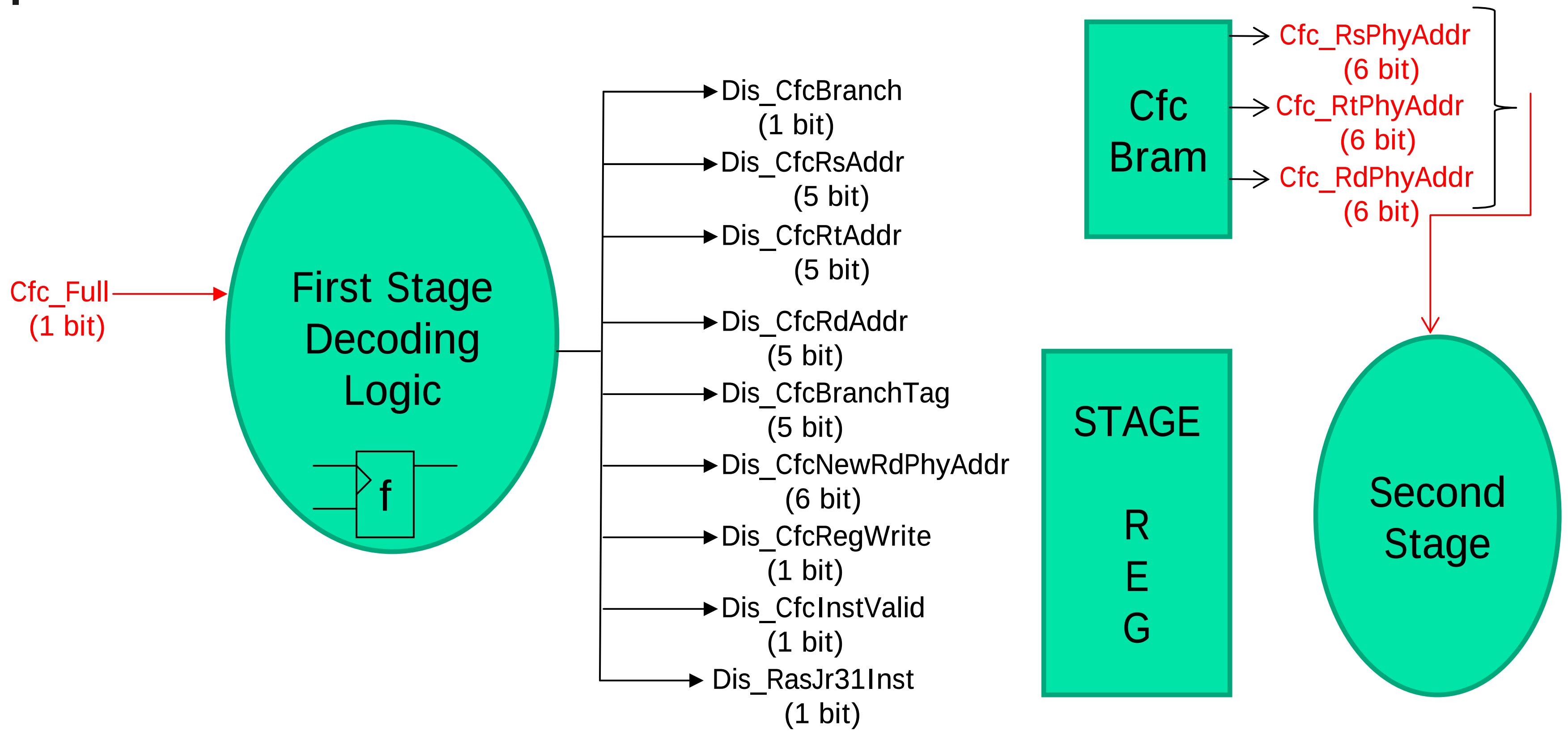
NOTE: The new available physical register from FRL is assigned to an instruction in the first stage only. This is harmless. In case the instruction gets flushed in second stage, FRL head pointer is restored as check-pointed by Cfc.

With CDB



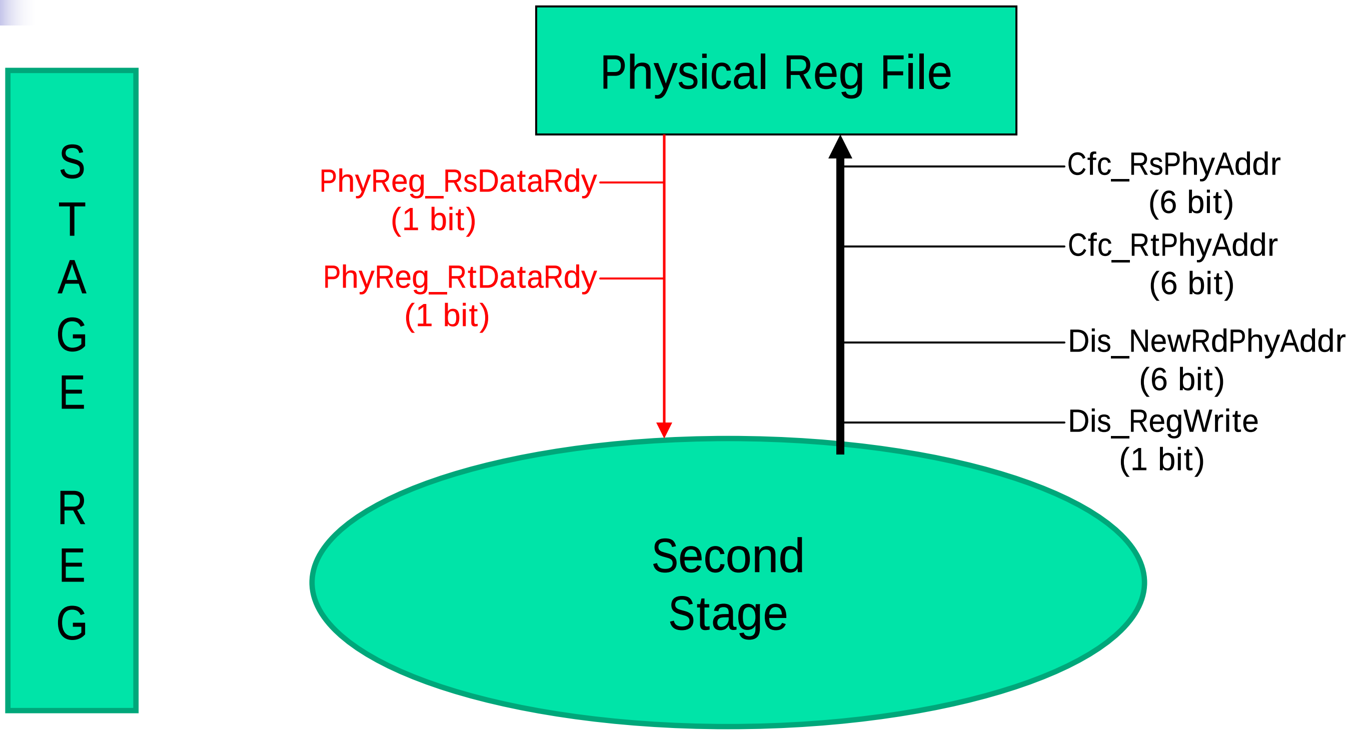
NOTE: Cdb_RobTag is needed in case of JR rs instruction. Dispatch needs to compare the RobTag of instrcution on Cdb with the tag of stalled JR RS to come out of stalling

With CFC



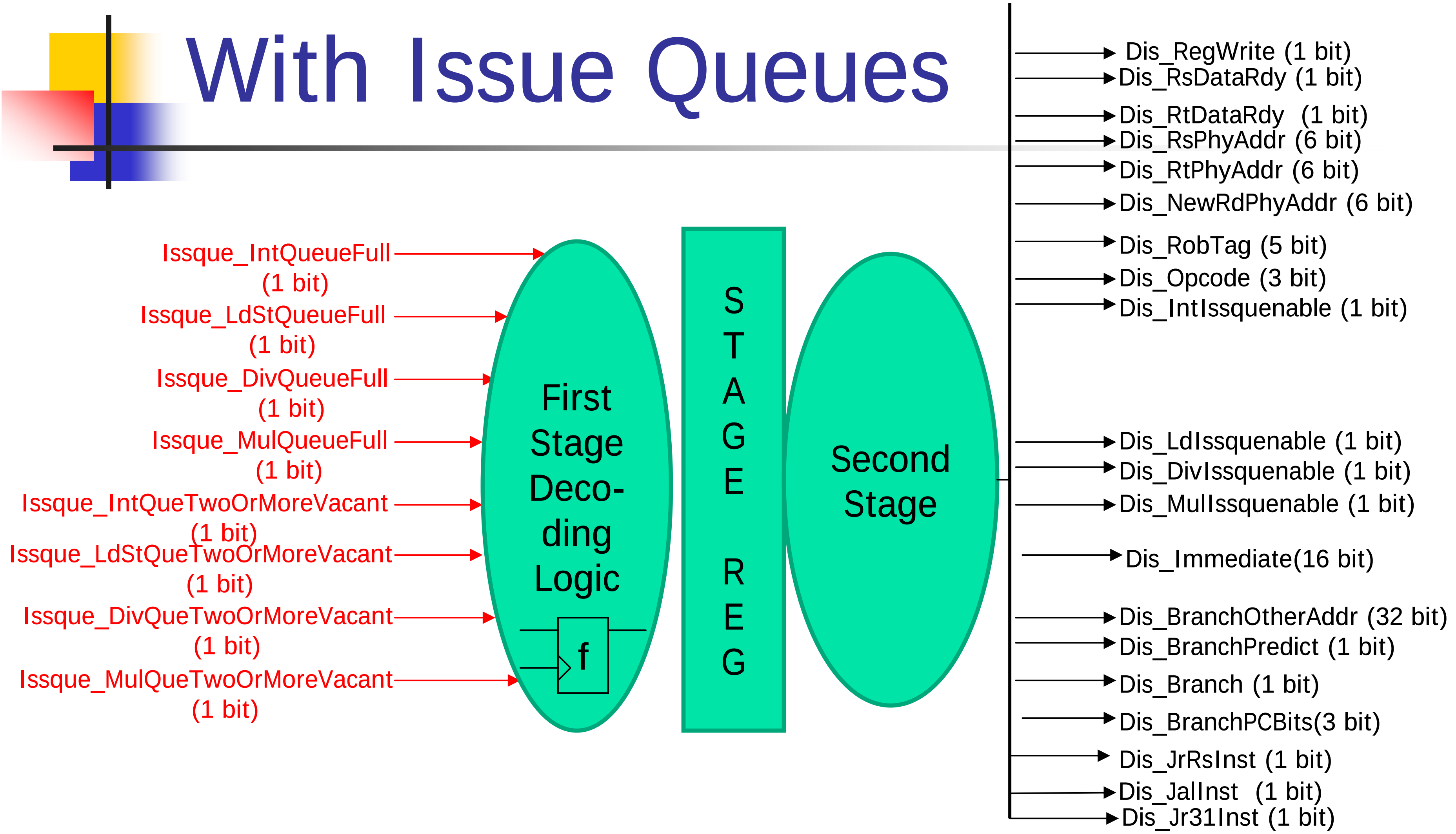
<u>NOTE:</u> The cfc unit needs **Dis_CfcBranch** and **Dis_RasJr31Inst** for check-pointing operation.
Dis_CfcRegWrite tells cfc unit that the instruction is register writing , thus active RAT needs to be updated

With Physical Register File



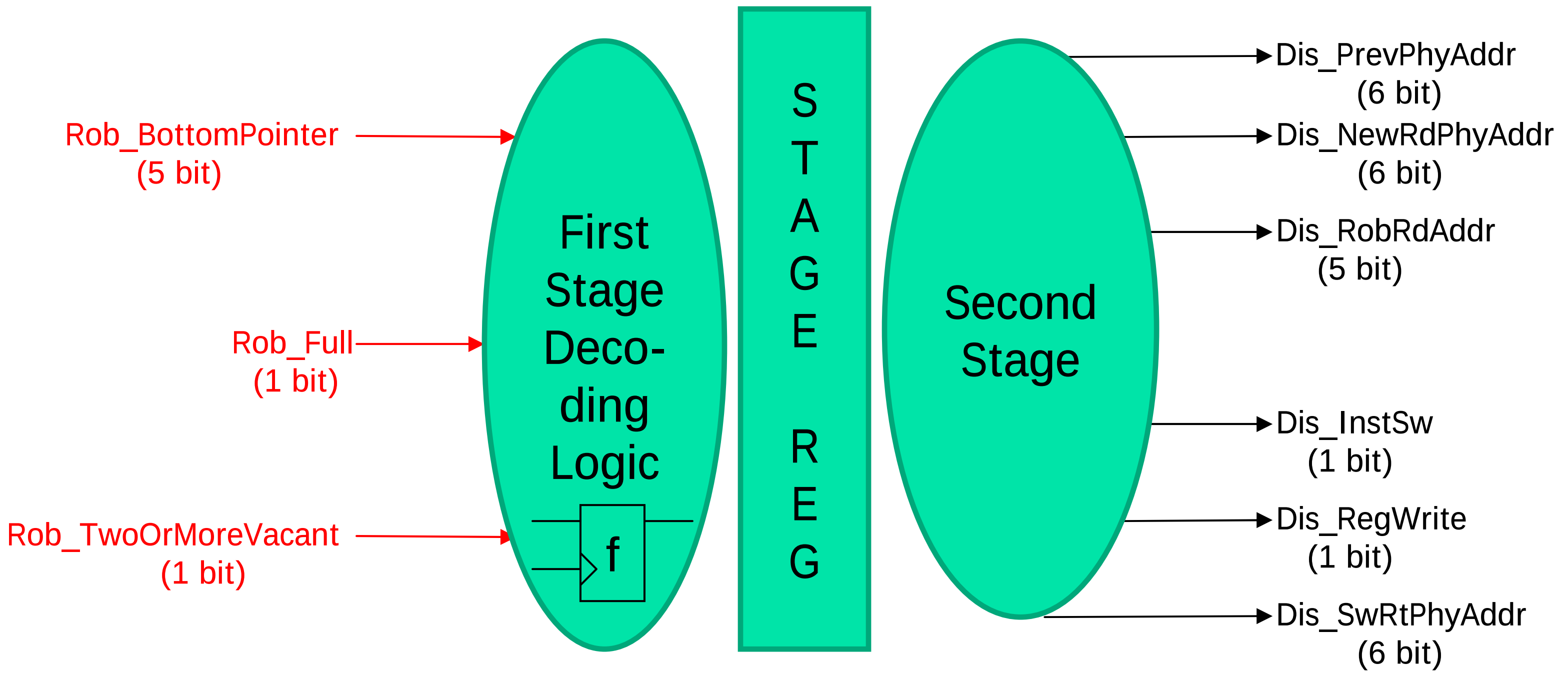
NOTE: Whenever a new physical register is assigned for “rd” register of an instruction , the physical register file needs to set the “ready” bit corresponding to that physical register to “0”. That is because on flushing we are not invalidating the ready bits of the physical registers which get freed (as it will call for invalidating multiple locations simultaneously). Thus **Dis_NewRdPhyAddr** tells the physical register to be assigned. **Dis_RegWrite** is needed to validate the operation as address pins may contain the address of used physical register when FRL is empty.

With Issue Queues


Issque_IntQueueFull (1 bit)
Issque_LdStQueueFull (1 bit)
Issque_DivQueueFull (1 bit)
Issque_MulQueueFull (1 bit)
Issque_IntQueTwoOrMoreVacant (1 bit)
Issque_LdStQueTwoOrMoreVacant (1 bit)
Issque_DivQueTwoOrMoreVacant (1 bit)
Issque_MulQueTwoOrMoreVacant (1 bit)
First Stage Deco-ding Logic
f
STAGE REG
Second Stage
Dis_RegWrite (1 bit)
Dis_RsDataRdy (1 bit)
Dis_RtDataRdy (1 bit)
Dis_RsPhyAddr (6 bit)
Dis_RtPhyAddr (6 bit)
Dis_NewRdPhyAddr (6 bit)
Dis_RobTag (5 bit)
Dis_Opcode (3 bit)
Dis_IntIssquenable (1 bit)
Dis_LdIssquenable (1 bit)
Dis_DivIssquenable (1 bit)
Dis_MulIssquenable (1 bit)
Dis_Immediate(16 bit)
Dis_BranchOtherAddr (32 bit)
Dis_BranchPredict (1 bit)
Dis_Branch (1 bit)
Dis_BranchPCBits(3 bit)
Dis_JrRsInst (1 bit)
Dis_JalInst (1 bit)
Dis_Jr31Inst (1 bit)

NOTE: Integer issue queue needs to distinguish between jr , jal and rest of the instructions. Thus **Dis_JrRsInst** (for jr $rs) **, Dis_JalInst** (for jal) and **Dis_JrInst** (for jr $31) are provided.

With ROB



NOTE: The “complete” bit for a store word instruction is set “1” as soon as its address is ready. The “ready bit” of physical register mapped to source (Rt) does not need to be checked. This is because the source register will be needed when sw reaches top of ROB. By that time all the instructions older to it will be committed and source register will be ready.