OoO Processor Design using HDL tools

EE560 Summer 2010 Project

Acknowledgement and recognition

EE 560 Summer 2010 Project Design Team

Six Directed Research students (and Prof. Gandhi Puvvada) in Spring 2010 have designed, proved it in simulation and implemented in FPGA

Manpreet Billing

Vaibhav Dhotre

Rajat Shah

Mohan SK

Atif Khan

Varun Khadilkar

Acknowledgement and recognition

EE 560 Summer 2010 Project Execution Team

Four Teaching Assistants (and Prof. Gandhi Puvvada) in Summer 2010 have refined the project slightly, designed Testbenches, simulation scripts, converted the project into design exercise for the EE560 class.

Prasanjeet Das

Waleed Dweik

Sabyasachi Ghosh

Da Cheng

Acknowledgement and recognition

EE 560 Summer 2010 Project has used nearly 50% of the **Summer 2009** project

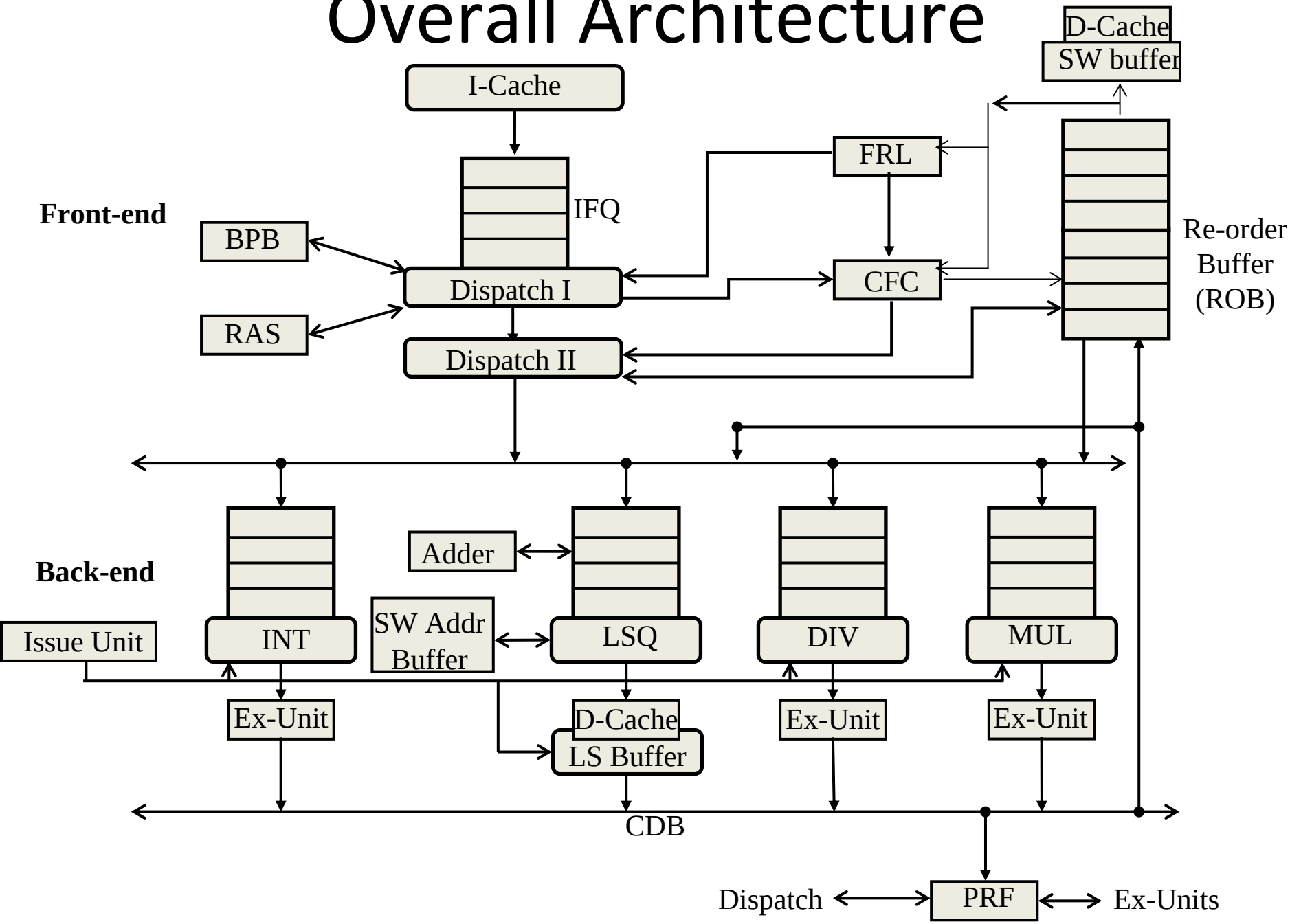
Contributors to the Summer 2009 Project

Spring 2009 Directed research team
Rohit Goel
Kapil Hede
Rajshekhar Ojha

Summer 2009 Teaching Assistants
Prasanjeet Das
Omair Rahman
Sukhun Kang

Special thanks to
Srinivas Vaduvatha for his contribution to the Summer 2008 project which led to the Summer 2009 and Summer 2010 projects!

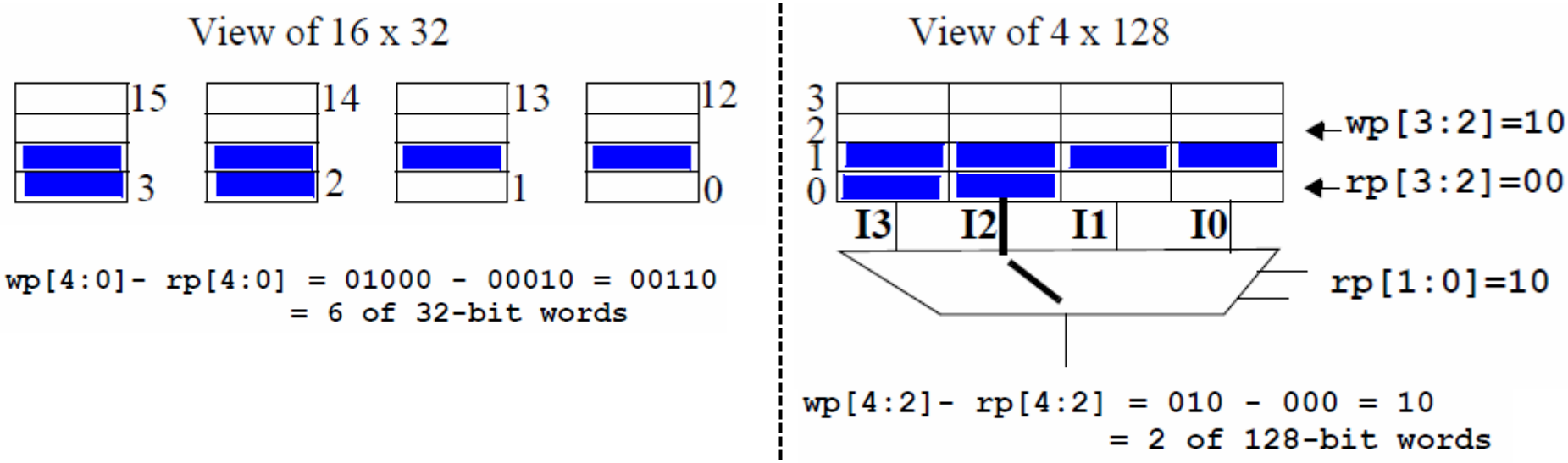
Overall Architecture



Overall Architecture

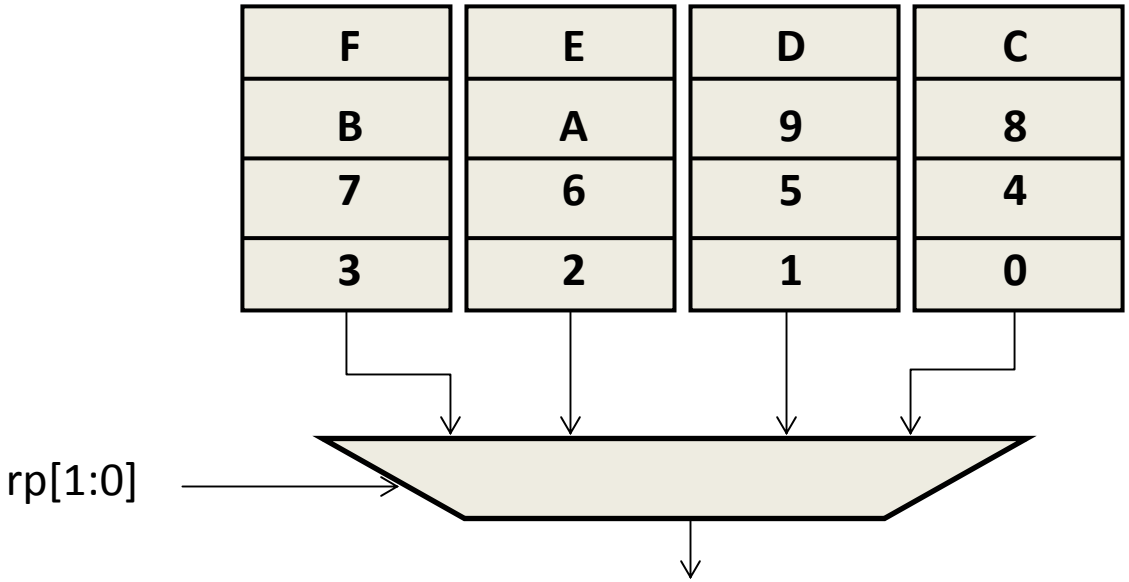
- Some of the important modules in the architecture:
  - IFQ (Instruction Fetch Queue, I_Cache)
  - Dispatch Unit -- Stage I and Stage II
  - FRL (Free Register List), RAS (Return Address Stack) and BPB (Branch Prediction Buffer)
  - CFC (Copy Free Checkpointing Module) (contains Ret. RAT)
  - PRF (Physical Register File)
  - Issue Queues and Ex. Units for Int, Mul, Div, Load-Store Address Buffer
  - Data Cache, Load Buffer, Issue Unit, CDB
  - ROB (Re-order Buffer), Store Buffer
- Instructions supported by Processor are:
  - Add, Addi, Sub, And, Or, Slt, Mul, Div
  - BNE, BEQ, J, Jal, Jr$31, Jr
  - LW, SW

Instruction Fetch Queue



- The Instruction Fetch Queue (IFQ) is a 16 location FIFO
- An entire cache line of 4 words is fetched from I-cache. So the FIFO in our design looks like a 4 x 128 for writing.
- Instructions are taken (one at a time) by the Dispatch Unit. So it looks like a 16 x 32 for reading.

Instruction Fetch Queue



- Four Way Interleaved fetching is done.
- IFQ is flushed every time Jump Instruction (or Jal or jr) comes in Dispatch or a Branch is predicted taken or a branch or jr $31 is mispredicted.

Dispatch

- At a time, an instruction can be in one of three states:
  - ❖ Fetched in the IFQ.
  - ❖ Incomplete, Pending in Backend.
  - ❖ Complete, Waiting in ROB, not committed
- Dispatch unit assigns a ROB slot to the instruction regardless of its type (including beq, bne, jal, and jr), except for jump or invalid instruction.
- For register writing instructions (except for $0 destination), Dispatch assigns a new physical register for the “rd” from the FRL (free register list). Also the Front-end RAT is searched for the previous register mapping, as the previous physical register assigned to this “rd” will be released when this instruction commits from the top of the ROB.
- For register Reading instructions, “rs” or “rs and rt” are searched in the Front-end RAT to find associated physical register and PRF is indexed to find Ready Bit for these source registers.

Dispatch

- Since Front-end RAT is implemented in CFC using BRAMs, the mapping is returned with 1 clock delay. This is why the dispatch unit here is a 2-stage unit.
- Since multiple instructions in back end can be writing to same architectural register , each destination register is given a ***new*** physical register. Every register reading instruction looks at its relevant physical register for architectural register value.

Branch Prediction and Handling

- Once Branch Instruction is decoded in dispatch, Branch Prediction is done using BPB (Branch Predictor Buffer).
- Bits PC[4:2] are used by the Branch Predictor to index BPB and if predicted taken, IFQ is flushed and new instructions are fetched from the computed Target Address.

Branch Prediction and Handling

- Branches are handled aggressively. They are executed from CDB instead from top of ROB. By that we mean that, we update BPB & if mis-predicted, flush wrong-path instructions and use the *other* address for fetching.

- Selective flushing mechanism is used to flush the instructions (younger to the mis-predicted branch or Jr instruction) in back end in case of mis-prediction.

Jumps (j, jal, jr)

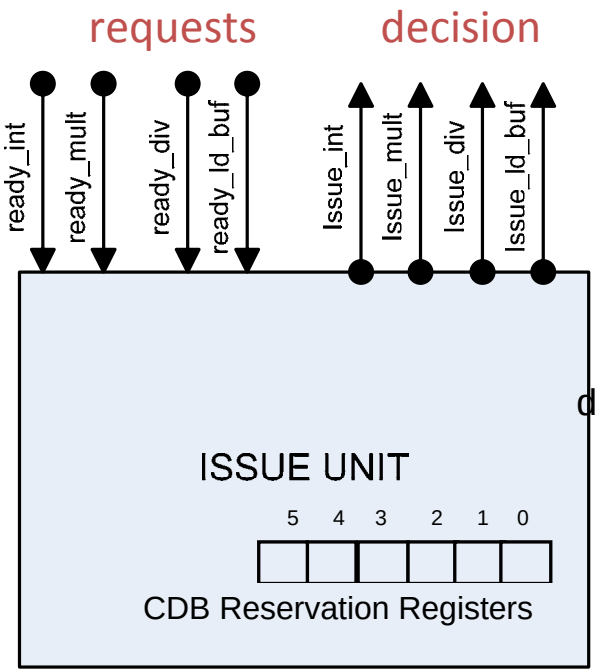
- Unconditional Jumps (j) are handled completely by dispatch itself.
- If instruction is decoded as JAL in dispatch, we push its PC+4 address on RAS and issue it to int queue as it is a register writing instruction (writing to $31) and start fetching from jump Address.

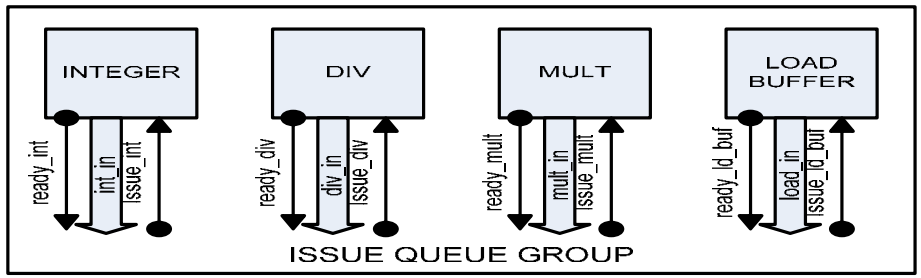
Jumps (j, jal, jr)

- If instruction is decoded as JR $31, we POP Return address from RAS and start fetching from that address. Since RAS can be corrupted, we consider the return address provided by RAS as a *prediction*. JR $31 (or JR $R) is issued to the Integer queue. If the actual contents of the $31 are different from the poped contents, JR$31 is declared as mispredicted.

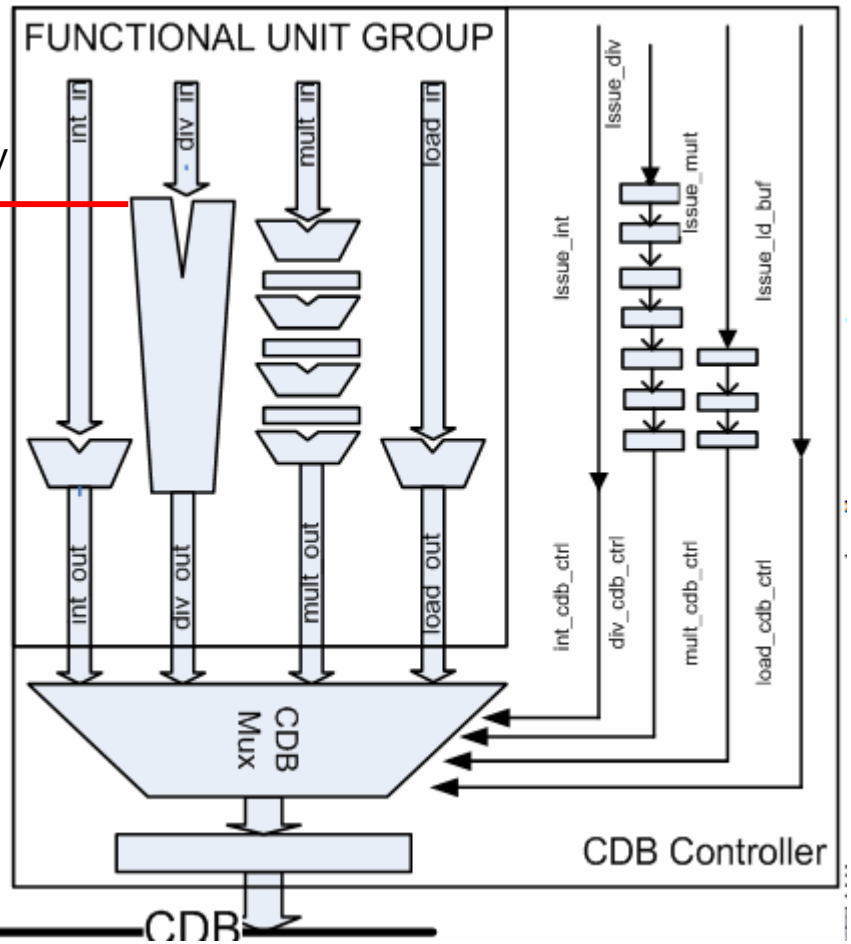
- In case of JR $R where R /= 31, we treat it as branch with no-prediction and stall till we execute and find target fetch addr.

Issue and Execute



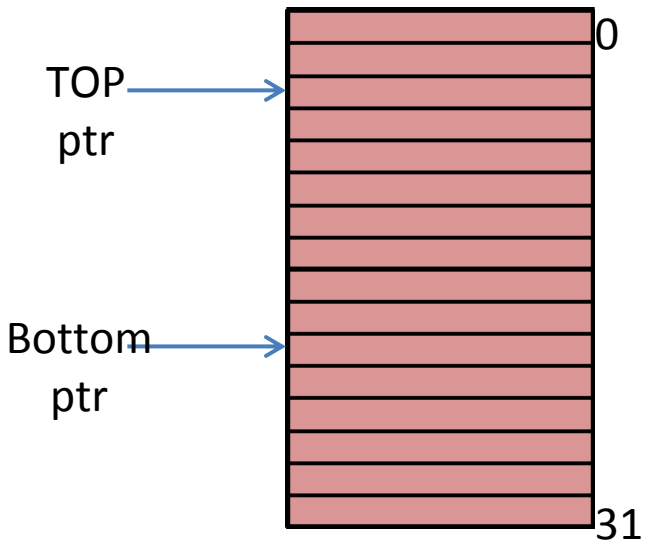


div_exec_ready



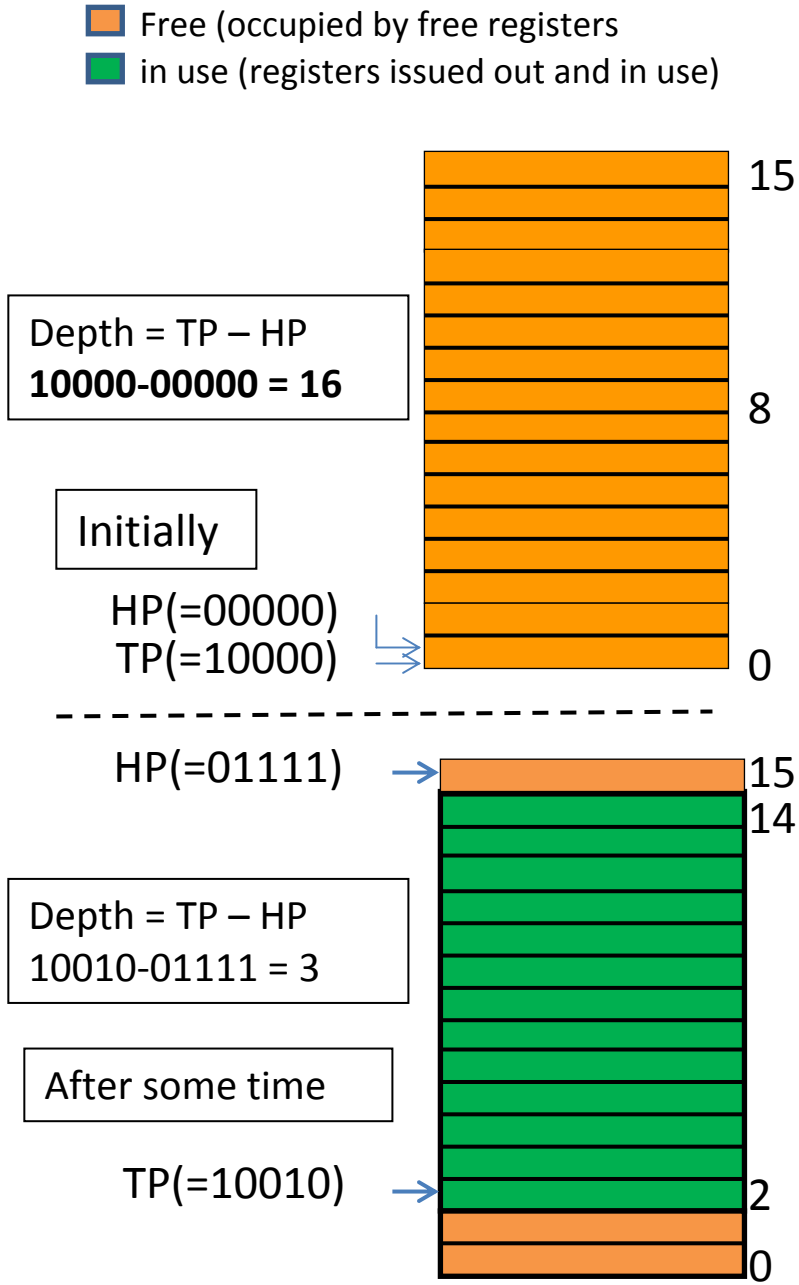
- Divider is Multi-Cycle Non-Pipelined
- Multiplier is Pipelined.
- LW accesses memory and the result is written in LS Buffer. SW goes directly to LS Buffer but it also updates “Store Addr Buffer”. When SW commits,we invalidate that entry.

ROB



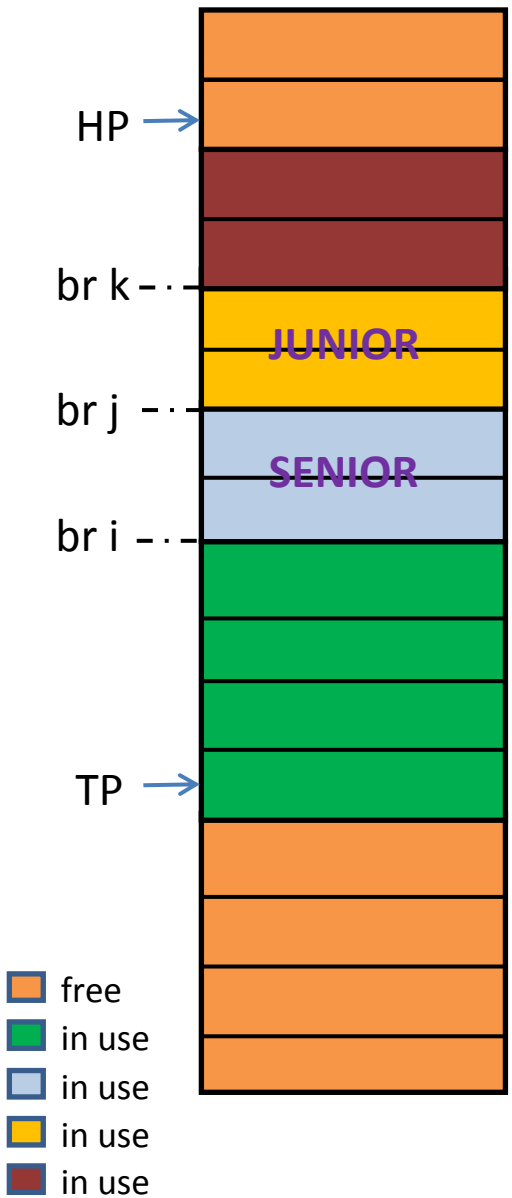
- Helps in In-Order Retirement.
- Instructions commit from the top of ROB.
- In case of mispredicted branch, to flush instructions from wrong path execution, we only need to adjust just the bottom pointer.
- In phase 2 of the project, ROB helps to restore CFC (in recovering after mis-predicted branch) through walking back or walking forward in ROB.

FRL



- FRL is a circular FIFO with 16 (=48-32) locations of 6 bit width referring to 16 of the 48 Physical Registers.
- Initially it is populated with 16 free registers (with PRF Ids 32 to 47 100000 to 101111). TP(WP) and HP(RP) are 5-bit pointers (not 4-bit). Initially, TP = 10000 and HP = 00000).
- As we dispatch, we use and assign registers from FRL by incrementing Head Pointer.
- As we Commit from Top of ROB, we free old (not new) physical registers to go back in FRL by incrementing Tail Pointer.

FRL



- From the example on the side, if the branch j is mispredicted, then the head pointer which was stored (in the stack) corresponding to branch j is restored back into the head pointer. This is what we do in phase 1 of our design.
- In phase 2 of our design, some branches have check-points and some do not. If branch j is not check-pointed, we jump to the nearest check-pointed branch (in ROB (and not in FRL)) and walk towards the mispredicted branch. We jump to branch i and walk forward (away from the top pointer in ROB emulating dispatch of intervening instructions and in FRL towards HP incrementing direction). Or jump to branch k and walk backward.
- While the above recovery is going on, the tail pointer can continue helping the registers which are freed on graduation of the instructions from the top of the ROB.

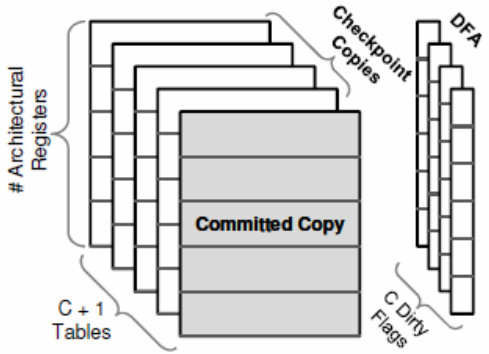
CFC Reference

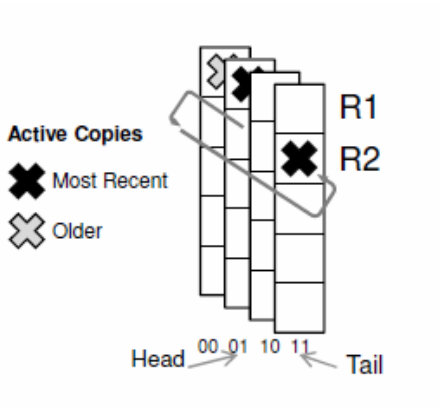
TOWARDS A VIABLE OUT-OF-ORDER SOFT CORE: COPY-FREE, CHECKPOINTED REGISTER RENAMING

Kaveh Aasaraai and Andreas Moshovos

Electrical and Computer Engineering
University of Toronto

Design ideas and figures are borrowed from the above paper





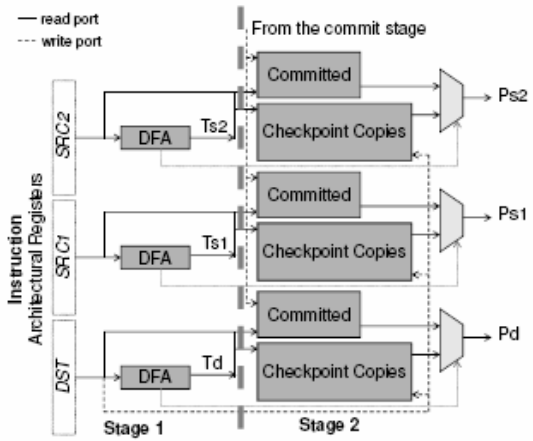
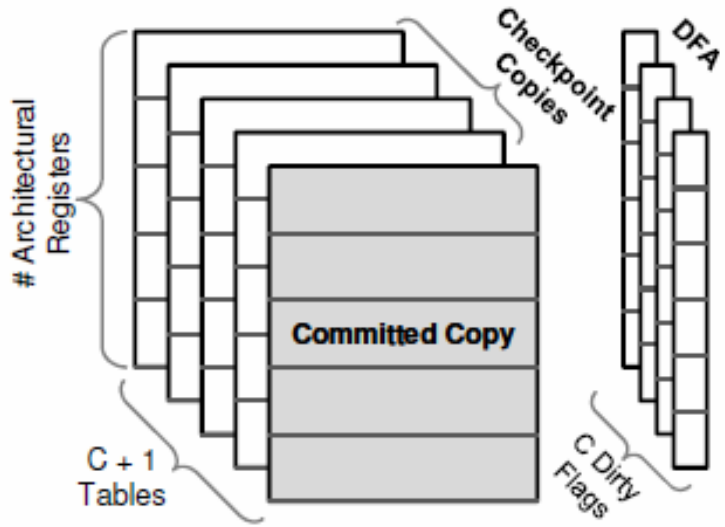


Fig. 6. CFC Implementation.

CFC



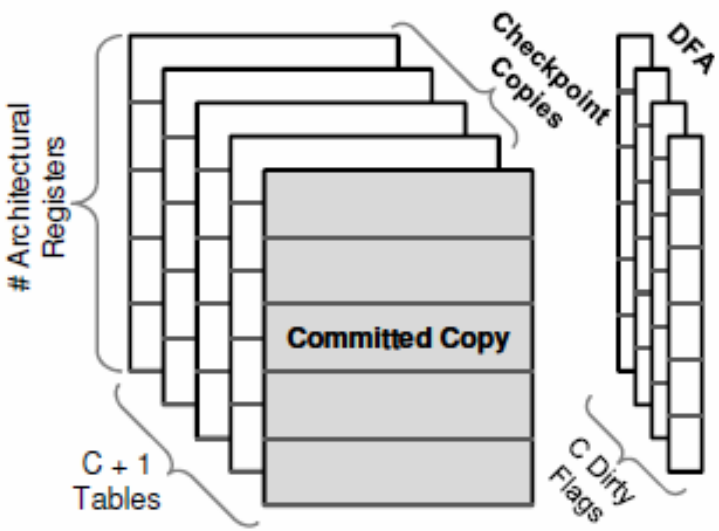
• Whenever a mapping of an architectural register in a register-writing instruction is done to physical register by dispatch unit, the corresponding DFA bit of the destination architectural register in the active table is set to 1.

• Instead of having the Front-end RAT, here we have the committed RAT (also called retirement RAT) and a set of incremental records of mappings as dispatch continues to dispatch.

• Consists of a committed RAT, 8 checkpoint RATs, and the associated 8 Dirty Flag Arrays (DFAs) (one for each RAT).

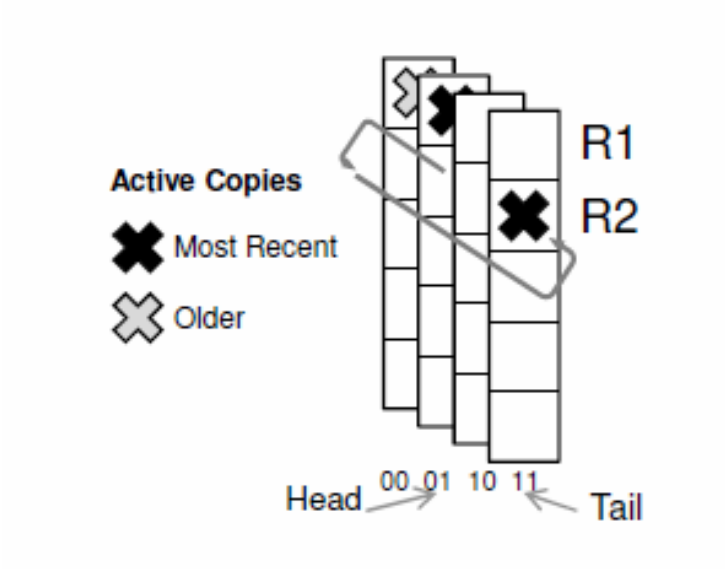
• The committed table (Retirement RAT) contains the latest committed mappings of architectural registers applied by non-speculative Instructions.

CFC



- When the system boots up, we will have the committed RAT and one active check point.
- As dispatch issues register writing instructions, the new physical register mappings are recorded in the active check point.
- When a branch instruction is dispatched, the active check point is frozen, and a new active check point is started.
- At any time, the Front end RAT, is represented *collectively* by the committed, Frozen, and Active RATs.

CFC – finding the latest mapping



- When dispatch asks for an architectural register mapping, say $2 (shown as R2 in the diagram on the side), the entire row of (note row, not column) of R2 (with index R2) is searched among the DFAs for the latest mapping. This search is a priority search with highest priority for the current active checkpoint (pointed to by the head pointer), going towards the oldest frozen checkpoint (pointed to by the tail pointer). If an entry is available, using that, the corresponding RAT is indexed to find the latest mapping.

- If no DFA bit is set for R2 row, then we go to the committed copy of the RAT for the latest mapping.

CFC

- Checkpoints are managed in a circular queue using two pointers – head & tail.

- The active table is denoted by the head pointer and the tail pointer indicates the table corresponding to oldest *uncommitted* checkpointed branch.

- Whenever a conditional branch or a JR $31 is dispatched, the active checkpoint is frozen, and a new fresh active check point is started.

CFC

- Whenever an instruction commits, the corresponding destination register mapping is written into committed table.
- As a result, a committed table always contains a valid RAT state that the processor can use to recover from any exceptional event.
- Whenever a check-pointed branch instruction commits, the tail pointer is moved ahead freeing the corresponding checkpointed table.
- Whenever the branch is mispredicted, the head pointer is simply moved to the checkpoint table created by the corresponding branch
  freeing 0 or 1 or more checkpoints.

CFC (for Phase 2)

- The latest value of Rs, Rd & Rt is provided to Dispatch by performing an associative search over corresponding DFA bit from head pointer to tail pointer. If none of the DFA bit is set, than the value is given from the committed copy like in phase 1.

- The mispredicted branch may not have been checkpointed due to lack of available free checkpoints or due to high confidence in the corresponding branch prediction.

- Then we jump to the nearest checkpoint preceding or following the failed branch (whichever is nearer) and walk forward or backward respectively. Correspondingly the head pointer is also shifted to that location in turn making that checkpoint as active.

CFC (for Phase 2) contd.

- Sometimes we may need to walk backwards from the current location in ROB without jumping to a previous checkpointed location.

- Sometimes we may need to walk forward all the way from the top of the ROB.