ECE401 PROJECT-2

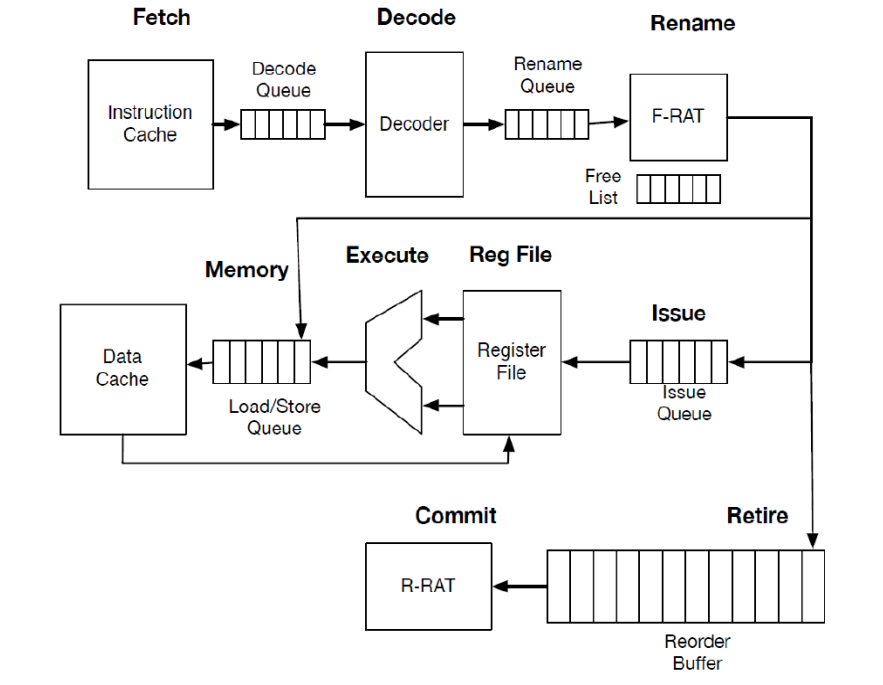
Submitted by
Chinmai
Nathan Whitehair

Aim of the project was to implement an out-of -order processor from an in order processor.

1. DESIGN REQUIREMENTS

To convert in-order pipeline processor to an out of order processor required several modifications and changes to the existing design. Some of the essential components required for implementation are as follows:

IF, ID, Issue/Commit	Specifications
Decode Queue	8 Entry FIFO
Rename Queue	8 Entry FIFO
Issue Queue	16 Entry
Load/Store Queue	16 Entry FIFO
Physical Register File	64 Entries
Commit	RRAT (32 Arch Registers)
Renaming	FRAT (32 Arch Registers)
Branch Prediction	Always not taken



3. IMPLEMENTATION

Converting existing in-order pipeline processor to out of order processor required us to remove forwarding and bypass logic of the pipeline processor. We also made several changes to the following stages:

FIFO.v:

This is a general module which maintains the data sent to it in a FIFO Q. It communicates with flags indicating if the FIFO is full or empty, also consists of DQ or NQ flags sent from any stage, if the respective stage wants insert or remove data from the FIFO.

Instruction Fetch & Decode Queue:

Fetches the instruction and Instruction PC from cache and inserts the fetched instruction to FIFO queue called Decode Queue. Fetch Stage waits for a feedback from Retire and commit, any time there is a branch mis-prediction. When there is a branch mis-prediction Decode queue is flushed and alternate PC sent from retire and commit stage is fetched by the IF stage. IF stalls when Decode queue is full and when there is a cache miss.

Instruction Decode & Rename Queue - This stage DQ's the data from Decode Queue and decodes each instruction sequentially. ID stages sends and NQ signal to rename queue whenever it wants to send data. ID stage stalls when Rename queue is full or when Decode queue is empty. Data sent to rename queue consists of Instruction, Instruction PC, architectural registers for destination and source register for the instruction and all the decode Flags.

Rename.v:

At this stage we have the decoded architectural registers. The destination register for every instruction is renamed with the first available free physical register from the Freelist. The busy bit for this Physical register is set and it is removed from the freelist. This stage sends in the data to Loadstore Queue, ROB and Issue Queue.

- FRAT- contains the architectural register mapping to the physical register.
- Freelist to maintain the free physical register list.
- BusyBits which maintains if the Physical registers are in use. The Busy bits are reset once the EXE has finished.
- Common Bus – this bus is received from retire and commit stage, once instruction is committed physical registers are free and sent back free list.

Issue Queue:

This is an out of order queue. The rename stage inserts into the Issue queue if the issue queue is empty.

- Wake up – Physical registers RS and RT are send to the issue queue from Rename stage along with the busy bits for each stage. If RS and RT registers are ready i.e if the busy bits are 0 then the instruction becomes ready.
- Selection – The very first instruction that is ready is sent to the Register read.
- Flush- Whenever there is branch misprediction, entire issue queue is flushed.

Register Read & EXE:

Registers received from the rename stage are then sent to Physical register file to obtain the operand values. The Register Read sends inn operands to EXE depending upon the flags each instruction.

OOOMEM.v

The OOOMEM module (Out-of-order Memory) contains an internal FIFO queue which serves as the Load-Store Queue. It handles load and store instructions. When it receives an instruction from the Rename stage, it enqueues it into the FIFO Load-Store Queue. When the ROB head is a load or store instruction, a signal is sent to OOOMEM, and the Load-Store head is committed, since it corresponds to the head of the ROB.

For a store instruction, this simply means sending a write request to the data cache, along with the data. We then dequeue the LSQ head and signal to the ROB that we are ready to commit another Load or Store if necessary. For a load instruction, we first send a read request to the data cache. We then wait for the request to be serviced; when it is, we write the data back to the register file and signal to the ROB that we are ready to commit another Load or Store if necessary. The ROB may not retire any other loads or stores while the OOOMEM is waiting for a cache read to complete; it waits until the transfer has finished.

OOOMEM also checks the common data bus for physical register activity; it any is detected, any instructions in the LSQ waiting for that data will be updated. The same thing happens when a writeback occurs from the OOOMEM stage.

RetireCommit.v

The RetireCommit module contains an internal FIFO queue which serves as the Reorder Buffer (ROB). Each ROB entry contains several fields describing different properties of each instruction. Entries are enqueued to it by the Rename stage.

When an instruction becomes the head of the ROB, several of its fields are evaluated:

- If the branch misprediction flag has been set (see below), we flush the ROB and the rest of the pipeline, then send the correct PC back to the IF stage so that it will retrieve the correct branch target. We also deliver the full contents of the RRAT back to the FRAT over a large data bus.
- If the load/store flag has been set, then we check to see if the OOOMEM stage is ready. If it is, we signal that it should commit the next instruction in the load/store queue. If the OOOMEM stage is not ready (i.e., it is waiting for a cache request to be serviced as described above), then we wait until the next clock edge and check again.
- If neither the branch misprediction flag nor the load/store flag has been set, or if the instruction is a load/store and the LS queue is ready, and the complete flag is set, then we dequeue the head of the ROB and overwrite the RRAT with the instruction's destination register alias.

The branch misprediction flag is set based on determinations made when a branch is resolved and its results are broadcast from earlier in the pipeline. When this happens, we compare the branch instructions UID to each instruction in the ROB. Upon finding a match we check to see if the branch was taken (speculatively) by the processor and compare this to whether the branch was meant to be taken. If there is a mismatch between these two, the misprediction flag is set. We also set the misprediction flag if the branch was taken but the wrong destination PC was used. (If the wrong destination was used but the branch was neither taken nor supposed to be taken, then the flag is not set.)

Note that our current implementation does not feature any dynamic branch prediction, however the features described above should allow for easy integration of a dynamic branch predictor (in terms of resolution and recovery).The 'complete' flag is set whenever an instruction has finished executing and is ready to retire/commit. The 'load/store' flag is set when the instruction is first enqueued to the ROB.

4. EXPERIMENTAL RESULTS

We were successfully able to send the data from the IF stage to rename stage. The rename stage successfully sends the data to Load Store Queue , ROB queue.

Figure 1 – Instruction PC shown below has the destination register as 29 and hence it is renamed with the first free physical register i.e 1 as shown in Figure 2.

Note – Physical register 0 is always used as an alias for architectural register 0.

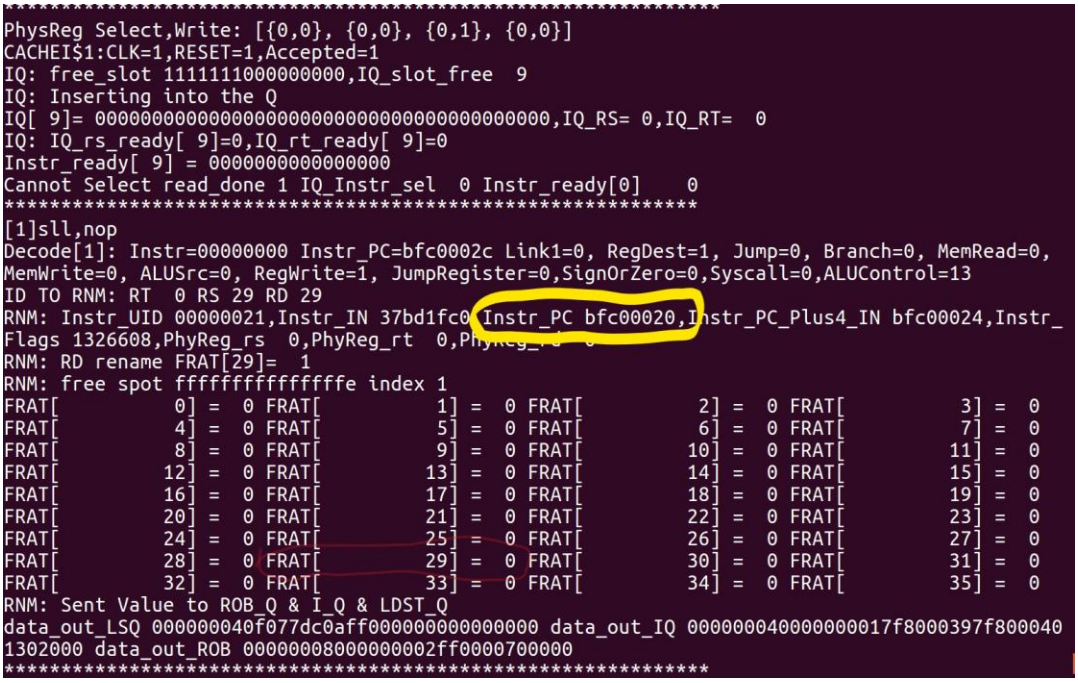


Figure 2

```
************************************************************
PhysReg Select,Write: [{0,0}, {0,0}, {0,1}, {0,0}]
CACHEI$1:CLK=1,RESET=1,Accepted=1
IQ: free_slot 1111110000000000,IQ_slot_free  9
IQ: Inserting into the Q
IQ[ 9]= 0000000400000000017f8000397f8000401302000,IQ_RS= 0,IQ_RT=  0
IQ: IQ_rs_ready[ 9]=0,IQ_rt_ready[ 9]=0
Instr_ready[ 9] = 0000000000000000
Cannot Select read_done 1 IQ_Instr_sel  0 Instr_ready[0]    0
************************************************************
[1]sll,nop
Decode[1]: Instr=00000000 Instr_PC=bfc00030 Link1=0, RegDest=1, Jump=0, Branch=0, MemRead=0,
MemWrite=0, ALUSrc=0, RegWrite=1, JumpRegister=0,SignOrZero=0,Syscall=0,ALUControl=13
ID TO RNM: RT  0 RS  0 RD  0
RNM: Instr_UID 00000022,Instr_IN 00000000,Instr_PC bfc00024,Instr_PC_Plus4_IN bfc00028,Instr_
Flags  311424,PhyReg_rs  0,PhyReg_rt  0,PhyReg_rd  0
RNM: RD rename FRAT[ 0]=  2
RNM: free spot ffffffffffffffc index 2
FRAT[          0] =  0 FRAT[          1] =  0 FRAT[          2] =  0 FRAT[          3] =  0
FRAT[          4] =  0 FRAT[          5] =  0 FRAT[          6] =  0 FRAT[          7] =  0
FRAT[          8] =  0 FRAT[          9] =  0 FRAT[         10] =  0 FRAT[         11] =  0
FRAT[         12] =  0 FRAT[         13] =  0 FRAT[         14] =  0 FRAT[         15] =  0
FRAT[         16] =  0 FRAT[         17] =  0 FRAT[         18] =  0 FRAT[         19] =  0
FRAT[         20] =  0 FRAT[         21] =  0 FRAT[         22] =  0 FRAT[         23] =  0
FRAT[         24] =  0 FRAT[         25] =  0 FRAT[         26] =  0 FRAT[         27] =  0
FRAT[         28] =  0 FRAT[         29] =  1 FRAT[         30] =  0 FRAT[         31] =  0
FRAT[         32] =  0 FRAT[         33] =  0 FRAT[         34] =  0 FRAT[         35] =  0
RNM: Sent Value to ROB_Q & I_Q
 data_out_IQ 0000000426f7a3f817f8000417f80004850f8400 data_out_ROB 000000084def47f02ff0000800
000
************************************************************
```

Figure 3 – Shows the freelist in rename stage, now that Physical register 1 is being used the next free physical register is 2 (index 2).

```
************************************************************
PhysReg Select,Write: [{0,0}, {0,0}, {0,1}, {0,0}]
CACHEI$1:CLK=1,RESET=1,Accepted=1
IQ: free_slot 1111110000000000,IQ_slot_free 10
IQ: Inserting into the Q
IQ[10]= 00000000000000000000000000000000000000000,IQ_RS= 0,IQ_RT=  0
IQ: IQ_rs_ready[10]=0,IQ_rt_ready[10]=0
Instr_ready[10] = 0000000000000000
Cannot Select read_done 1 IQ_Instr_sel  0 Instr_ready[0]    0
************************************************************
[1]sll,nop
Decode[1]: Instr=00000000 Instr_PC=bfc00034 Link1=0, RegDest=1, Jump=0, Branch=0, MemRead=
MemWrite=0, ALUSrc=0, RegWrite=1, JumpRegister=0,SignOrZero=0,Syscall=0,ALUControl=13
ID TO RNM: RT  0 RS  0 RD  0
RNM: Instr_UID 00000023,Instr_IN 00000000,Instr_PC bfc00028,Instr_PC_Plus4_IN bfc0002c,Ins
Flags  311424,PhyReg_rs  0,PhyReg_rt  0,PhyReg_rd  0
RNM: RD rename FRAT[ 0]=  2
RNM: free spot 11111111111111111111111111111111111111111111111111111111111111100 index 2
FRAT[          0] =  0 FRAT[          1] =  0 FRAT[          2] =  0 FRAT[          3] =
FRAT[          4] =  0 FRAT[          5] =  0 FRAT[          6] =  0 FRAT[          7] =
FRAT[          8] =  0 FRAT[          9] =  0 FRAT[         10] =  0 FRAT[         11] =
FRAT[         12] =  0 FRAT[         13] =  0 FRAT[         14] =  0 FRAT[         15] =
FRAT[         16] =  0 FRAT[         17] =  0 FRAT[         18] =  0 FRAT[         19] =
FRAT[         20] =  0 FRAT[         21] =  0 FRAT[         22] =  0 FRAT[         23] =
FRAT[         24] =  0 FRAT[         25] =  0 FRAT[         26] =  0 FRAT[         27] =
FRAT[         28] =  0 FRAT[         29] =  1 FRAT[         30] =  0 FRAT[         31] =
FRAT[         32] =  0 FRAT[         33] =  0 FRAT[         34] =  0 FRAT[         35] =
RNM: Sent Value to ROB_Q & I_Q
 data_out_IQ 0000000440000000017f8000497f8000501302000 data_out_ROB 00000008800000002ff0000
000
************************************************************
EXE:Instr1=00000000,Instr1_PC=00000000,ALU_result1=00000000; Write?0 to  0
EXE:ALU_Control1=00; MemRead1=0; MemWrite1=0 (Data:00000000)
EXE:OpA1=00000000; OpB1=00000000; HI=00000000; LO=00000000
```

Figure 4: Destination Register for the next instruction is set with the free physical register.

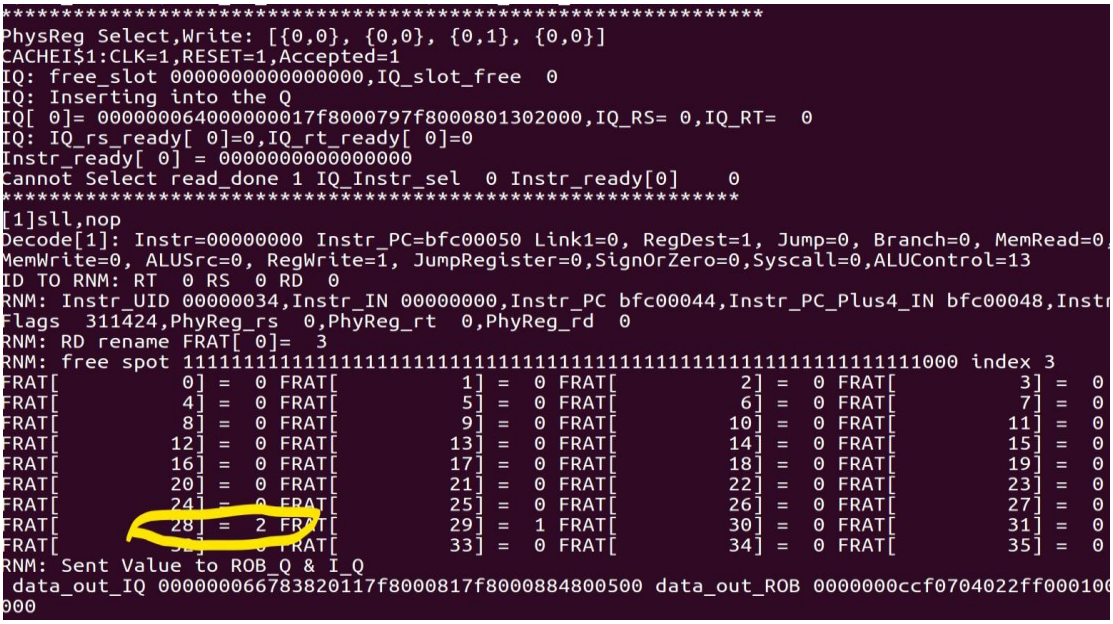


Figure 5: Depicts how the instruction is fetched from cache and inserted into the IDQUEUE(decode queue) at the same time few of the instruction are enqueueing and dequeueing from Load store, ROB and rename queue.