EE739A - Advanced Processor Design
Project I : Superscalar IITB RISC

Meet Udeshi - 14D070007
OV Shashank - 14D070021
Arka Sadhu - 140070011

May 4, 2017

1 Fetch

1.1 Program Counter

- This register is the speculative program counter which is used to fetch from the instruction cache
- The actual permanent PC is stored in the ARF corresponding to register R7 as R7 == PC in the ISA
- It is updated by PC+2 every cycle, unless a branch misprediction correction is issued or the fetch stage is stalled

1.2 Branch Predictor (BP)

- Stores Target Address and Prediction History corresponding to BEQ, JAL and JLR. Though JLR is a multi-taregted branch, assuming that it would be used much as a multi-target, we predict it using a single BTA. The other R7 write instructions could also be predicted with minimal hardware change, but to prevent BP Pollution we are not including them.
- JLR is not stores becuase its target address needs to always be computed and can be multitargeted which adds issues in checking whether the branch taken was to the right address or not
- Branches are by default assumed to be not taken so that PC+2 rule can be continually followed
- Planned to be implemented as a two bit predictor because the ISA does not allow from good prediction accuracy with small hardware

2 Decode

2.1 Decoders

- There are two parallel decoders which generate the necessary control signals for the future stages. This also includes the necessary reordering of the operands into source and destination registers
- They include necessary hardware to detect inter-dependencies between the two instructions fetched
- They generate inter-dependency bits which are later to be used in the renaming stage by the ARF and RRF to correctly generate and provide the tags
- The validity of the destination operand for the ARF as well as for the carry and zero flags are produced which is passed on as the invalid tag bits in the execution stage

2.2 LM/SM Handling Block

- It is responsible for replacing the decode block in case of arrival of an LM or SM
- It generates the necessary operands and addresses that are to be written to or read from. This is performed by generating signals similar to the load instruction and simply by incrementing the immediate field
- To ensure that the value read from Address Source Register does not change an isLM bit is generated which is a signal for harware in the dispatch stange which ensures correct execution of the instruction

3 Dispatch

3.1 Register Renaming

Registers are 16-bit. Carry and Zero Flag registers are 1-bit. Renaming is performed for both the registers individually.

3.1.1 Architectural Register File

- They store tags of the corresponding rename regiser for every Architectural registers, along with non-speculative data.
- It will also have a valid bit. Whenever an interrupt or flush in the ROB occurs all valid bits are set to 1.
- If the AR is renamed, then it is invalid, else it is valid.
- It is set valid, only when the tag pointed by it, and the tag broadcasted by the ROB matches.
- It will take into account the interdependency bits and use the RRF queue to decide which tags to issue and which to be provided to the reservation station in case of a intra RAW hazard.

3.1.2 Rename Register File

- Execution will give the tag for the RR, and the corresponding register will update its value.
- This will also be broadcasted RS.

3.1.3 RRF Queue

- Queue of available rename registers, updated based on ROB tag broadcast.

3.2 Reservation Station

The reservation station is maintained as a hybrid b/w the distributed and central system. One reservation station handles the ALU operations and the other handles memory based instructions. The policies for each are different and they are explained below

3.2.1 Allocation Policy

- Arithmetic (Branch) RS
  - Keep a queue (circular), to store the available RS entries for allocation.
  - This has to check whether or not to stall.
  - The register values are from the pipeline registers, valid bits from the ARF, and ROB tag from ROB.
  - It also allocates to the ROB, and checks for stalls.

- Memory (Branch) RS
  - For the memory instructions becuase they are supposed to executed in order becuase we do not have load bypassing or load dependency checks, we must excute the loads and the stores strictly in order
  - Hence the allocation is performed like in a FIFO, new entries are inserted after the last tail entry

3.2.2 Issue Policy

- Arithmetic (Branch) RS
  - Top-down search for ready bits.
  - Decides between the two ALU pipes and sends NOP(s) if (an) instruction(s) is/are not available
- Memory (Branch) RS
- Issue is performed only if the the top queue entry is ready. A NOP is issued otherwise.
- This is again necessary to maintain and loads and stores in order

3.2.3 Table Entries

- Source Rename Register Tags.
- Source Register Values and Immediate Data
- Destination Rename Register Tags
- ROB tag
- Control Signals including invalidTag bits

3.2.4 LM/SM Handling Additions

- These additions are only made to the memory RS. This is not duplicated for every entry but is only one of its kind in the entire RS
- Registers are necessary to store the tag as well as the value of the starting address of the LM/SM instruction
- Storage of tag is necessary if the Address generating instruction (AGI) has not yet cleared the busy bit. This allows for the read to occur when the address is generated by the execution units
- Storing the value is necessary because, once the AGI exits the ROB, the register tag stored would enter the RRF Queue and the value may be replaced which would lead to the remaining LM instructions reading the wrong instructions. And hence once the value has been written to the RRF, it is simultaneously written to this value store register (VSR) and a valid bit alongside is set to say that the value in the VSR is valid and that the tag must not be read anymore.

4 Execution

4.1 3 Pipes

The execution unit has one memory pipe which also handles the memory based R7 write branches and two arithmetic and branch pipes

4.2 Arithemetic/Branch Pipes

5 Write Back

5.1 Reorder Buffer (ROB)

The re-order buffer is implemented as a circular buffer, with head and tail pointers. The head pointer helps in deciding which instructions to retire and the comparison of the head and tail pointer decide when to call for a stall when the ROB get full.

Every cycle 4 of the ROB Tags are set to finished based on the 4 bits which signify the validity of the tags (for example a tag is invalid if it was NOP instruction)

5.1.1 Table Entries

- Instruction Address
- Destination Tag of Flags and Register
- Invalid Tag Bits for Flags and Register
- Speculative Bit and Valid Bit
- Finished Bit

5.2 Store Queue

To allow for lazy write. Cannot write to memory in the execution stage