CS-204

Design Document : RISC-V 32I Simulator

Phase - 2

Group Number : 3

Ayush Verma 2019CSB1147
Bhumika 2019CSB1152
Keshav Krishna 2019CSB1224
Rishabh Jain 2019CSB1286
Vishwam Datta 2019CSB1305

Hazard Unit:-

This file(hazard.py) basically contains the hazard unit, which basically detects any form of data hazard with previous instructions and informs the control unit about what data forwarding to do , how many stalls etc.All this unit takes as input is the opcode,rd ,rs1 ,rs2 ,funct3,funct7 of the current instruction.
The previous two instructions are stored in the queue. The current instruction dependency (Control hazard or data hazard) is checked with each of the previous two instructions.
The decision-maker function calls check_dependence_function which detects the dependency and returns two integers in a list according to the type of dependency between current instruction and previous-to-previous instruction and with previous instruction respectively.

Important Methods:-
Based on whether the forward knob is set or not two functions are called which are:

- Data_forwarding(i1,i2)
    - ➢ Arguments -> 2 integers i1 and i2.i1 tells the type of dependence b/w current instruction and previous to previous instruction, i2 tells the type of dependence b/w current instruction and previous instruction.
    - ➢ Return Value-> Now based on the values of i1 and i2, 2 Integers would be returned [a,b] a->encoding of forwarding b/w i1 and i3
      b->encoding of forwarding b/w i2 and i3
      This encoding gives information to the control circuit about what kind of forwarding is to be performed from the previous 2 instructions to the current instructions if any , and also which stage is to be stalled if any stalls are needed.

-1: No forwarding
0: M to M (No stall) every time forward to rs2
11: M to E (No stall) to rs1 only
12: M to E (No stall) to rs2 only
13: M to E (No stall) to both rs1 and rs2
21: M to E (1 stall of E)to rs1 only
22: M to E (1 stall of E) to rs2 only
23: M to E (1 stall of E)to both rs1 and rs2
31: E to E (No stall) to rs1 only
32: E to E (No stall) to rs2 only
33: E to E (No stall) to both rs1 and rs2
41: M to D (Decode is stalled 2 times) to rs1 only
42: M to D (Decode is stalled 2 times) to rs2 only
43: M to D (Decode is stalled 2 times) to both rs1 and rs2
51: M to D (Decode is stalled 1 time) to rs1 only
52: M to D (Decode is stalled 1 time) to rs2 only
53: M to D (Decode is stalled 1 time) to both rs1 and rs2
61: E to D (1 stall of D) to rs1 only
62: E to D (1 stall of D) to rs2 only
63: E to D (1 stall of D) to both rs1 and rs2
71: E to D (no stall) to rs1 only
72: E to D (no stall) to rs2 only
73: E to D (no stall) to both rs1 and rs2

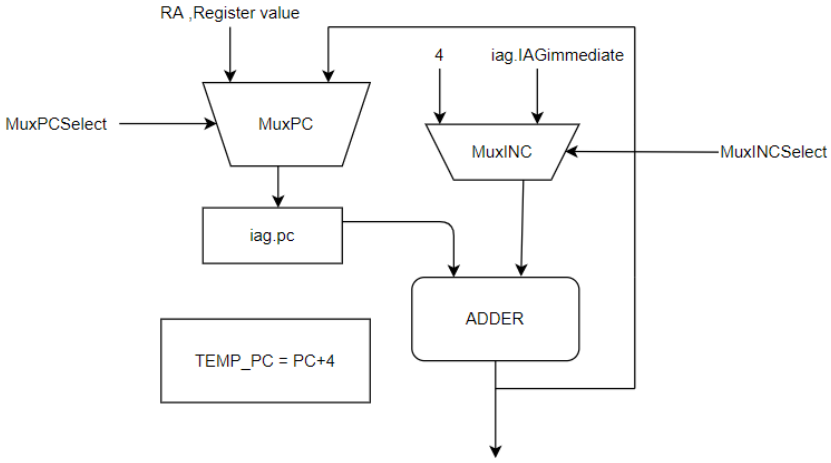
- data_stalling(i1,i2)
  - ➢ Arguments -> 2 integers i1 and i2.i1 tells the type of dependence b/w current instruction and previous to the previous instruction, i2 tells the type of dependence b/w current instruction and previous instruction.
  - ➢ Return Value-> Now based on the values of i1 and i2, 1 Integer would be returned which tells information about how many cycles should the decode stage be stalled.

  The encoding of the return is as follows:
  -1:No stalling
  1: Stall D for 1 cycles
  2: Stall D for 2 cycles

- decision_maker(opcode,funct3,rs1,rs2,rd,forwarding_knob)
  Based on whether the forwarding knob is 1 (which signifies data forwarding is enabled) or 0 (which signifies data forwarding is disabled) this function makes the appropriate calls to either data_forwarding() (this returns a tuple of encoding telling what kind of action is to be performed, i.e type of data forwarding and stalling) function or data_stalling() function (how many stalls), this is done after first identifying what kind of dependence exists between the current instruction and previous instruction or previous

to previous instruction or both if any and then returns the decision made to the control unit which then would proceed to perform these forwarding and stalling operations.

IAG Module: -



Inputs:-

- MuxINCselect - Control signal of MuxINC
- MuxPCselect - Control signal of MuxPC
- Boffset - To set the IAG value.

Output:-

- PC_temp - To store this value for jal and jalr statement as return address in a register.

Important Variables:-

- PC - SImulates the PC register in IAG.
- PC_temp- Simulates the PC_temp register in IAG
- IAGimmediate- Simulates the immediate input given to MuxINC

Methods:-

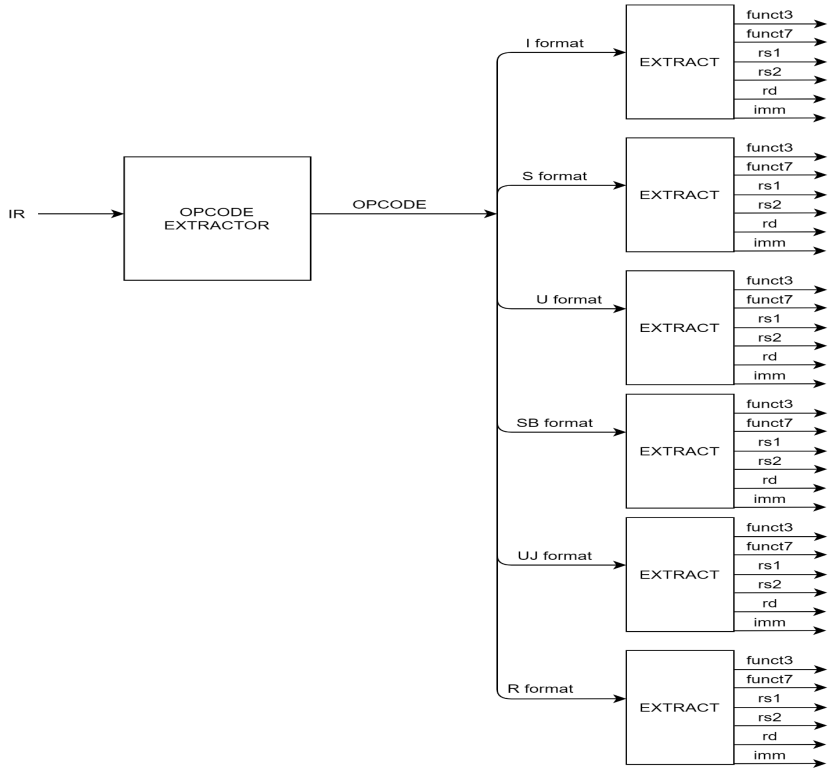
- ReadPC - Used to get the current value of PC
- WritePC - Used to update/ write the value of PC
- PCTempUpdate - Used to set the value of PC_temp to PC + 4
- PCset - Used to select , based on control signals whether to put PC as RA (register input in case of Jalr) or let PC remain PC.
- SetBranchOffset - Used to update the immediate value that is connected to MuxINC
- PCUpdate- Based on the control signals that are given as arguments , this function decides whether to set PC as PC + 4 or PC + immediate.
- BTB_check - check wether this instruction (PC) already exists in the table or not
- BTB_insert - This will insert the PC and the destination instruction address (if the branch is taken) in the BTB and the prediction corresponding to it , whether the branch is to be taken or not.The values in this table is in the form of a BTB_entry class variable which

contains the variables ->inst_address, target_address and take (which signifies whether to take the branch or not , hence is a boolean value)

Decode Unit :-
Input: The input to the Decode unit is IR (instruction in hexadecimal format) and PC.
The opcode of the machine instruction is calculated using bit masking.
If the opcode equals “0x11” (17), the program gets terminated.
Based on the opcode of the machine instruction, following functions are called which extract the respective fields (rs1, rs2, rd, function 3, function 7 and immediate value) from the respective instruction format :

- decodeR # for R format instruction (add, and, or, sll, slt, sra, srl, sub, xor, mul, div, rem)
- decodeI # for I format instruction (addi, andi, ori, lb, lh, lw, jalr)
- decodeS # for S format instruction (sb, sw, sh)
- decodeSB # for SB format instruction (beq, bne, bge, blt)
- decodeU # for U format instruction (auipc, lui)
- decodeUJ # for UJ format instruction (jal)

In each of the above functions, bit masking is used to extract the fields from the machine code.
The immediate field is taken special care, as it is a signed value. Signed value is ensured by use of ‘&’ and ‘|’ logical operators.
Output: Control signal generator function is called by the decode unit after extracting the fields from the machine instructions.



Control signals

1. Control signals for the instructions will be generated in the decode stage and saved(enqueued) in the respective control signal queues of each of the stages.
2. Each stage will dequeue the control signals from the queue before execution.
3. The queues will be initialized with an appropriate offset.
4. Appropriate NOPs will be passed for bubbles.
5. Exclusive for Fetch and Decode- These stages will have a queue each. If queue==empty, then operate. If not empty then NOP condition. Pop the element and return from the function.(this is since these stages don't have any specific control signal except NOP)
6. In data forwarding, we can push a control signal in the stage to take data forwarding from a particular module when it is enqueued.