Performance Modelling - RISC-V processor

This project will require you to implement cycle-accurate simulators of a 32-bit RISC-V processor in C++ or Python. The skeleton code for the assignment is given in file (NYU_RV32I_6913.cpp or NYU_RV32I_6913.py).

The simulators should take in two files as inputs: imem.text and dmem.txt files
The simulator should give out the following:

- cycle by cycle state of the register file (RFOutput.txt)
- Cycle by cycle microarchitectural state of the machine (StateResult.txt)
- Resulting dmem data after the execution of the program (DmemResult.txt)

The imem.txt file is used [illegible] used to initialize the data memory of the pro[illegible]ruction or the data memory and both the ir[illegible] that for a 32 bit processor, 4 lines in the [illegible]ta memory are in "Big-Endian" format (th[illegible]

The instructions to be s[illegible]types:

Bit	*31*	*7, 6*	*0*
R-type	funct7 (7 bits		Opcode (7 bits)
I-type	imm		Opcode
S-type	imm[11:5]		Opcode
B-type	imm[12, 10:5		Opcode
U-type			Opcode
J-type			Opcode

The simulator should support the following set of instructions.

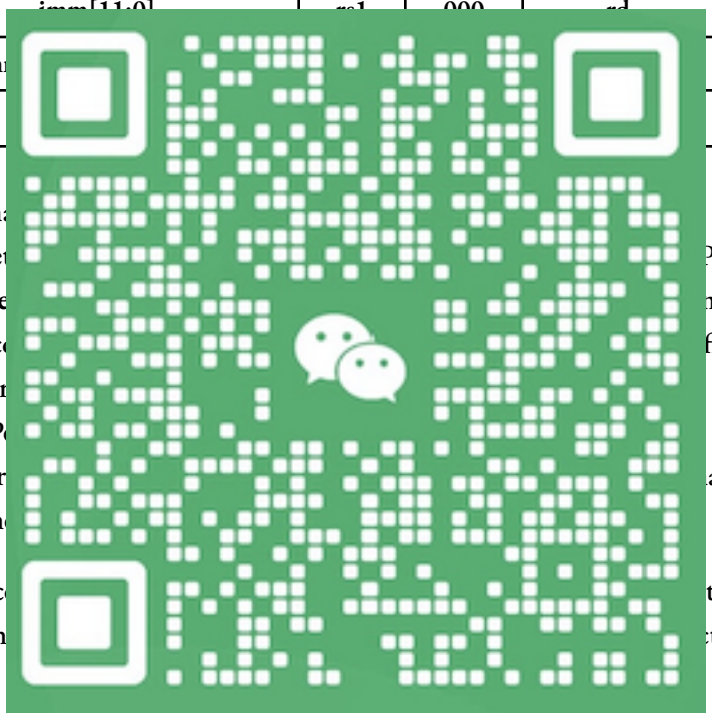
Mnemonic	Type	Full Name	Psuedocode	Details
ADD	R	Addition	rd = rs1 + rs2	Store the result of rs1 + rs2 in register rd.
SUB	R	Subtraction	rd = rs1 - rs2	Store the result of rs1 - rs2 in register rd.
XOR	R	Bitwise XOR	rd = rs1 ^ rs2	Store the result of rs1 ^ rs2 in register rd.
OR	R	Bitwise OR	rd = rs1 | rs2	Store the result of rs1 | rs2 in register rd.

AND	R	Bitwise AND	rd = rs1 & rs2	Store the result of rs1 & rs2 in register rd.
ADDI	I	Add Immediate	rd = rs1 + sign_ext(imm)	Add the sign-extended immediate to register rs1 and store in rd. Overflow bits ignored.
XORI	I	XOR Immediate	rd = rs1 ^ sign_ext(imm)	Bitwise XOR the sign-extended immediate to register rs1 and store result in rd.
ORI	I	OR Immediate	rd = rs1 | sign_ext(imm)	Bitwise OR the sign-extended immediate to register rs1 and store result in rd.
ANDI	I	AND Immediate	rd = rs1 & sign_ext(imm)	Bitwise AND the sign-extended immediate to register rs1 and store result in rd.
JAL	J	Ju		C + sign_ext(imm) and store the n rd.
BEQ	B	Br		(PC = PC + sign_ext(imm)) if rs1 is
BNE	B	Br		(PC = PC + sign_ext(imm)) if rs1 is
LW	I	Lo		e at memory address [rs1 + nd store it in rd.
SW	S	St		of rs2 to memory address [rs1 value].
HALT	-	Halt execution		

Instruction encoding:

Mnemonic	Bit Fields						
	31:27	26:25	24:20	19:15	14:12	11:7	6:0
ADD	0000000		rs2	rs1	000	rd	0110011
SUB	0100000		rs2	rs1	000	rd	0110011
XOR	0000000		rs2	rs1	100	rd	0110011

OR	0000000	rs2	rs1	110	rd	0110011
AND	0000000	rs2	rs1	111	rd	0110011
ADDI	imm[11:0]		rs1	000	rd	0010011
XORI	imm[11:0]		rs1	100	rd	0010011
ORI	imm[11:0]		rs1	110	rd	0010011
ANDI	imm[11:0]		rs1	111	rd	0010011
JAL	imm[20|10:1|11|19:12]				rd	1101111
BEQ	imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011
BNE	imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011
LW						0000011
SW						0100011
HALT						1111111



The simulator should ha

- **Instruction Fe** PC value as address.
- **Instruction De** mat in the table above and generates c file.
- **Execute:** Perfo
- **Load/ Store:** P
- **Writeback:** Wr at R0 in RISC-V can only contain th

Each stage must be prec to the next stage in the next cycle. Each stage sh tive in the following cycle.

The simulator must be able to deal with two types of hazards.

1. RAW Hazards: RAW hazards are dealt with using either only forwarding (if possible) or, if not, using stalling + forwarding. Use EX-ID forwarding and MEM-ID forwarding appropriately.
2. Control Flow Hazards: The branch conditions are resolved in the ID/RF stage of the pipeline.

The simulator deals with branch instructions as follows:

1. Branches are always assumed to be NOT TAKEN. That is, when a beq is fetched in the IF stage, the PC is speculatively updated as PC+4.
2. Branch conditions are resolved in the ID/RF stage.

3. If the branch is determined to be not taken in the ID/RF stage (as predicted), then the pipeline proceeds without disruptions. If the branch is determined to be taken, then the speculatively fetched instruction is discarded and the nop bit is set for the ID/RR stage for the next cycle. Then the new instruction is fetched in the next cycle using the new branch PC address.

Tasks:

1) Draw the schematic for a single stage processor and fill in your code in the to run the simulator. (20 points)
2) Draw the schematic for a five stage pipelined processor and fill in your code to run the simulator. The processor should be able ot take care of RAW and control hazards by stalling and forwarding. (20 points)
3) Measure and report ... le for both these cores by adding performa... console or a file.) (5 points)
4) Compare the result... ssor implementations and explain why on...
5) What optimization... dit 1 point)

Your work will be evalu... week before the deadline. (50 points - 5 ...

Useful References:

- More details on... https://riscv.org...
- bitset library fo...
- g++: https://gc... C.html
- python: https://www.python.org/downloads/