Design Doc: Detailing the structural and logical design of your program

Details of logical design:

Our task:

1. **Implement Caches.**
2. **Pipeline modifications.**

Cache informations:
• I-cache: 8 KiB, 16 byte block size, 4-bit offset, 9-bit index, 19-bit tag bits
8KiB / 16 bytes = 512 blocks (# of lines)

• D-cache: 16 KiB, 8 byte block size, 3-bit offset, 11-bit index, 18-bit tag bits, write-through
16KiB / 8 bytes = 2048 blocks (# of lines)

Implement Caches:

- .We use struct to represent a cache. For I-cache, the struct includes members: valid bit, tag and 4 instructions(since the block size is 16 bytes).
- For D-cache, the struct includes members: valid bit, tag and data.(Since the block size is 8 bytes)

For each cache, we have a set of functions (check and update):

1. check_i_cache:
   We throw the PC address into the function, dividing the address into 3 parts(offset, index, tag).
   - If the corresponding cache's tag matches the pc's tag and the valid bit =1, we return an array result containing status and instruction.(hit)
   - If the tags are matched but valid bits = 0(miss), we return an array result. Tag and valid bits need to be updated.
   - If the tags do not match each other, we return an array result. Tag and valid bits need to be updated

2. update_i_cache:
   This function updates the tag and instructions in $I. We throw the PC address into the function, dividing the address into 2 parts(offset, tag).
   - Update tag
   - Update instructions
   - Update valid bit.

3. check_d_cache:
   We throw the destination address into the function, dividing the address into 2 parts(index, tag).
   - If the corresponding cache's tag matches the destination address's tag and the valid bit =1, we return an array containing status and data and offset.(hit)

- If the tags are matched but valid bits = 0(miss), we return an array result. Tag and valid bits need to be updated.
- If the tags do not match each other, we return an array result. Tag and valid bits need to be updated.

4. Update_d_cache:
   This function updates the tag and instructions in $D. We throw the destination address into the function, dividing the address into 2 parts(index, tag).
   - Update tag
   - Update data
   - Update valid bit.

5. write_d_cache:
   This function check if the write to $D miss or hit.
   - If it's write hit, then update the value in the d-cache
   - If it's write miss, then leave the d-cache unmodified.

Pipeline modifications:

In order to notify the fetch stage to stall, we implement two booleans in the stage_reg_d struct which are i_cache_stall and first_cache_stall.

1. Fetch:
   For the first time we fetch values from memory to I-cache through memory_ read. If the boolean i_cache is true, we stall.
   After the first time we read from memory status. If we can read something, there is no need to stall. Otherwise we stall.

- Access cache functions
- in F&M stages
- When memory is not ready ---> stall and fetch.
- Write hit: D-cache updated, write value to mem.
  Write miss: D-cache unchanged, write value to mem
- Fetch: only memory reads,
  Memory: memory reads and write.

Task assignment

- Hang Yuan
    - The basic structure of the program
    - No. 20-35 sub-functions in execution stage
    - Debugging
- Zhewei Wang
    - Memory stage
    - No. 36-51 sub-functions in execution stage
    - Debugging
    - Testing
- Andrew Tsai
    - Register stage
    - No. 1-19 sub-functions in execution stage
    - Debugging

o	lb 1
o	lh 2
o	lw 3
o	ld 4
o	lbu 5
o	lhu 6
o	lwu 7
(X)	fence 8
(X)	fence.i 9
o	addi 10
o	slli 11
o	slti 12
o	sltiu 13
o	xori 14
o	srli 15
	srai 16
o	ori 17
o	andi 18
	auipc 19
	addiw 20
	slliw 21
	srliw 22
	sraiw 23
o	sb 24
	sh 25
	sw 26
	sd 27
	add 28

	sub 29
	sll 30
	slt 31
	sltu 32
	xor 33
	srl 34
	sra 35
	or 36
o	and 37
o	lui 38
o	addw 39
o	subw 40
	sllw 41
	srlw 42
	sraw 43
	beq 44
	bne 45
	blt 46
	bge 47
	bltu 48
	bgeu 49
	jalr 50
	jal 51
(X)	ecall 52
(X)	ebreak 53
(X)	csrrw 54
(X)	csrrs 55
(X)	csrrc 56
(X)	csrrwi 57
(X)	csrrsi 58
(X)	csrrci 59
🟩	mul 60
	div 61
	rem 62