CS2610: Computer Organization and Architecture

Scalar Pipelined Processor Design

Submission Deadline: 16^{th} Apr, 2022

- Objective: To understand the working of scalar pipelined processor.
- Team Size: 2

1 Processor Configuration

Consider a scalar pipelined processor that has a 256B instruction cache (I$) and a 256B data cache (D$), both having a read port and a write port each and both are direct-mapped caches (the block size is 4B). Assume that both instruction and data caches are perfect, meaning there won't be any cache misses in these caches. Assume that the processor has a register file (RF) with sixteen, 8-bit registers, named R0, ..., R15. Note that R0 always stores the value "0". The register file has two read ports and a write port. Negative numbers are stored in 2's complement form.

2 The Pipelined Processor

2.1 The Basic Pipeline

We consider a *Reduced Instruction Set Computer* (RISC) processor. The RISC architecture has the following instruction set.

The instruction set of the processor includes

- Four arithmetic instructions:
  ```
  ADD R1, R2, R3      ; R1 = R2 + R3
  SUB R1, R2, R3      ; R1 = R2 - R3
  MUL R1, R2, R3      ; R1 = R2 * R3
  INC R1      ; R1 = R1 +1
  ```
- Three logical instructions:
  ```
  AND R1, R2, R3      ; R1 = R2 & R3, bit-wise AND
  OR  R1, R2, R3      ; R1 = R2 | R3, bit-wise OR
  NOT R1, R2          ; R1 = ∼ R2; 1's complement; Here last four bits are discarded.
  XOR R1, R2, R3      ; R1 = R2 ⊕ R3, bit-wise XOR
  ```
- Two data transfer instructions:
  ```
  LOAD R1, R2, X      ; R1= [R2+X]
  STORE R1, R2, X     ; [R2+X] = R1
  ```
- Two control transfer instructions:
  ```
  JMP L1              ;Unconditional jump to L1; Last four bits are discarded.
  BEQZ R1, L1         ;Jump to L1 if R1 content is zero.
  ```
  L1 is given as an offset from current Program Counter (PC). This is called PC-relative addressing. L1 can be an 8-bit number represented in 2's complement format.

- Halt instruction

  `HLT` ;Program terminates; Least significant 12 bits are discarded.

We consider a five-stage instruction pipeline: IF-ID-EX-MEM-WB. For a store instruction, the WB stage is non-existent. For ALU instructions, the MEM stage is non-existent. The processor is pipelined at the instruction level. The instructions are assumed to be of fixed length of 2 bytes each.

1. ***Instruction Fetch Cycle*** (IF):

   `IR ⟵ I$[PC];`

   `PC ⟵ PC + 2;`

   Operation: Send out the Program Counter (PC) and fetch the instruction from the instruction cache (IL1 cache) into the Instruction Register (IR); increment the PC by 2 to address the next sequential instruction. The IR is used to hold the instruction that will be needed on subsequent clock cycles; Register PC is updated to point to the address of the next instruction. The above describes fetching of one instruction at a time. You should fetch one instruction at any time in the scalar pipeline architecture.

2. ***Instruction Decode Cycle*** (ID):

   Operation: Decode the instruction; identify the opcode, source registers, and a destination register; establish the input/output dependency information. Since there are 16 registers, 4 bit encoding is used for registers. R0 is encoded as 0000, R1 as 0001, etc. All the fields are at a fixed location in the instruction format. 4-bit opcode is considered in the instructions as shown below:

Opcode	4-bit Representation	Opcode	4-bit Representation
ADD	0000	SUB	0001
MUL	0010	INC	0011
AND	0100	OR	0101
NOT	0110	XOR	0111
LOAD	1000	STORE	1001
JMP	1010	BEQZ	1011
HLT	1111		

   In the case of LOAD/STORE instructions, the least significant four bits provide the offset, which can be a positive/negative number.

   In this pipeline stage, we also read source operands from the RF and the outputs of the general purpose registers are read into two temporary registers (A and B) for use in later clock cycles.

   ```
   A ⟵ RF[R2];
   B ⟵ RF[R3];
   ```

3. ***Execution/Effective Address Cycle*** (EX):

   The ALU operates on the operands prepared in the prior cycle, performing one of the following three functions depending on the instruction type.

- **Memory Reference: (LD and ST)**
  `ALUOutput ⟵ R2 + X;`
  Operation: The ALU adds X with the contents of R2 fetched in earlier cycle to form the effective address and places the result into the temporary register `ALUOutput`.
- **Register-Register ALU Instruction:**
  `ALUOutput ⟵ A op B`
  Operation: The ALU performs the operation specified by the opcode on the values in registers A and B. The result is placed in the temporary register `ALUOutput`.
- **Branch:**
  `ALUOutput ⟵ PC + (L2 << 1);`
  `Cond ⟵ (A == 0)`
  Operation : The ALU adds the PC to the sign-extended immediate value in `L2`, which is shifted left by 1 bit to create a 2 byte offset, to compute the address of the branch target. Register A, which has been read in the prior cycle, is checked to determine whether the branch is taken. Since we are considering only one form of branch (`BEQZ`), the comparison is against 0.

4. ***Memory Access Cycle*** (MEM):
   `LMD ⟵ D$[ALUOutput]` or
   `D$[ALUOutput] ⟵ B;`
   Operation: Access data cache, if needed. If the instruction is a load, data returns from the data cache and is placed in the LMD (load memory data) register; if it is a store, the data from the B register is written into the data cache. In either case the address used is the one computed during the prior cycle and stored in the register `ALUOutput`.

5. ***Write-Back Cycle*** (WB):
   - Register-Register ALU Instruction:
     `RF[R1] ⟵ ALUOutput;`
   - Load Instruction:
     `RF[R1] ⟵ LMD;`
     Operation: Write the result into the register file, whether it comes from the memory system (which is in `LMD`) or from the ALU (which is in `ALUOutput`).

2.2 Pipelining Hazards

Hazards are situations that prevent the next instruction in the instruction stream from getting executed in its designated clock cycle. Hazards may stall the pipeline. We consider data hazards and control hazards.

- **Read-After-Write (RAW) Hazards**:
  Consider the instruction sequence given below.

```
ADD R1, R2, R3
SUB R4, R1, R5
```

The content of R1, which is produced by the ADD instruction, is required for the SUB instruction to proceed.

- **Control Hazards**: Arise from pipelining of branches and other instructions that change the Program Counter (PC). For example, in a conditional Jump instruction, till the condition is evaluated, the new PC can take either the incremented PC value or the address accessed in that instruction. To avoid this we stall the pipeline for 2 cycles.

When a conflict is encountered, all the instructions before the stalled instructions need to continue and all the instructions after the stalled instruction need to be stalled.

3 Deliverables

- **Input**: Three files: ICache.txt (representing the contents of Instruction Cache), DCache.txt (representing the contents of Data Cache), and RF.txt (representing the contents of Register File).
- **Output**: Two files: DCache.txt (it should reflect all modifications to the data in Data Cache because of executing an application) and Output.txt (containing the important statistics as described below).
- Assuming that each pipeline stage takes 1 cycle, execute a given program (specified in the machine language) and report 1) the total number of instructions executed; 2) the total number of instructions for each type; 3) the CPI (clock cycles per instruction), 4) the number of stalls, and 5) the reason for each stall (ex: RAW dependency).
- Interested students can implement *operand forwarding* in their design for extra credit.