Y86-64 Sequential Implementation

Team Members: -

Akshit Gureja (2020112004),

Atharva Sunil Gogate (2020112001)

Contents

Overview

The aim of the project is to implement a modular Y86-64 ISA design for processor architecture in Verilog with a 5-stage pipelined implementation. The first part of the project sees us approaching the problem statement with a sequential implementation (without pipelining), which is what this report elaborates on.

Sequential Design

Fetch

The fetch stage requires us to read an instruction from the instruction memory and find—

- `icode`
- `ifun,`
- `rA,`
- `rB,`
- `valC`

The instruction memory is initialised as an array of registers.

The updated PC value is received on the positive edge of the clock and looks at the next instruction to be read from the instruction memory. If PC goes beyond the maximum allowed value, we get a `memory_error`. A valid PC value results in the PC byte being taken and `icode` and `ifun` being obtained. Based on `icode`, which tells us which instruction to execute—

- 0 remaining bytes are read if the instruction is halt, nop or ret,
- 1 byte immediately after the PC byte is read if the instruction is cmovXX, OPq, pushq popq,
- 9 consective bytes after the PC byte are read if the instruction is irmovq, rmmovq or mrmovq,
- 8 consective bytes after the PC byte are read if the instruction is jXX or call.

Based on the bytes, we can obtain (if required according to the instruction), rA, rB, valC.

- halt, nop, ret: valP = PC + 64’d1
- cmovXX, OP1, pushq, popq: valP = PC + 64’d2

- irmovq, rmmovq, mrmovq, jXX, call: valP = PC + 64’d10

Decode

The decode stage requires us to assign appropriate values to valA and valB depending on what icode, rA, rB are.

The values to be assigned to valA and valB are stored in the register numbers rA and rB.

reg_memory is a length-15 array of 64-bit registers and represents the register memory.

valA = reg_memory[rA] for cmmovq, rmmovq, OPq, pushq

valB = reg_memory[rB] for rmmovq, OPq, mrmovq,

valA = reg_memory[4] for ret, popq

valB = reg_memory[4] for call, ret, pushq, popq

Execute

The ALU sees extensive use by the execute stage.

valE is assigned the appropriate value based on icode and ifun.

ifun values are of much importance especially in the case of cmovXX and jXX, where it decides the condition of the conditional move or jump.

Memory

This stage is used for reading and writing to the memory. The memory is declared separate from the instruction memory and does not see use in the other stages as it is accessed only in this stage. The icode value decides whether we read from and/or write to memory.

rmmovq, call, pushq: write to memory

mrmovq, ret, popq: read from memory

The other instructions do not require us to read from or write to memory.

Write back

This stage is used to write up to two results to the register file.

PC Update

We utilise this stage to either set PC to the address of the next instruction or simply valP.

Testing

Testing of stages has been done.

Testing individual instructions is not fully completed but we are very close to achieving it. However, testing status codes (AOK, HLT, INS) still remains.

Challenges Encountered

Defining instruction memory and register memory proved to be quite a challenge, as was defining their indexing. Additionally, calling 2-dimensional register arrays in other files is not possible and a workaround for that had to be thought of by dividing the contents of such arrays into individual registers.