Pipelined RISC CPU Project

This project aims to create a staged pipeline processor running a RISC ISA. The processor also handles hazards to provide efficient performance.

A harvard architecture (different code and data memories) is assumed for memory.

The processor is specified using verilog behavioral models.

Instructions' format of the design

Memory Instructions

Instruction	Op-Code	First Operand	Second Operand
LDM R_dst, Imm	10010	R_dst2 R_dst1 R_dst0	I7 I6 I5 I4 I3 I2 I1 I0
LDD R_src, R_dst	10011	R_dst2 R_dst1 R_dst0	R_src2 R_src1 R_src0
STD R_src, R_dst	10000	R_dst2 R_dst1 R_dst0	R_src2 R_src1 R_src0
PUSH R_dst	10100	R_dst2 R_dst1 R_dst0	xxxxxxxx
POP R_dst	10111	R_dst2 R_dst1 R_dst0	xxxxxxxx

Branch Instructions

Instruction	Op-Code	First Operand	Second Operand
JZ R_dst	11000	R_dst2 R_dst1 R_dst0	xxxxxxxx
JN R_dst	11001	R_dst2 R_dst1 R_dst0	xxxxxxxx
JC R_dst	11010	R_dst2 R_dst1 R_dst0	xxxxxxxx
JMP R_dst	11011	R_dst2 R_dst1 R_dst0	xxxxxxxx

ALU With Immediate

Instruction	Op-Code	First Operand	Second Operand
SHL R_dst	11110	R_dst2 R_dst1 R_dst0	I7 I6 I5 I4 I3 I2 I1 I0
SHR R_dst	11111	R_dst2 R_dst1 R_dst0	I7 I6 I5 I4 I3 I2 I1 I0

Port Instructions

Instruction	Op-Code	First Operand	Second Operand
IN R_dst	11100	R_dst2 R_dst1 R_dst0	xxxxxxxx
OUT R_dst	11001	R_dst2 R_dst1 R_dst0	xxxxxxxx

Special Instructions

Instruction	Op-Code	First Operand	Second Operand
NOP	00000	xxxxxxxx	xxxxxxxx
SETC	00111	xxxxxxxx	xxxxxxxx
CLRC	00110	xxxxxxxx	xxxxxxxx
CALL R_dst	00101	R_dst2 R_dst1 R_dst0	xxxxxxxx
RET	00010	xxxxxxxx	xxxxxxxx

Instruction	Op-Code	First Operand	Second Operand
RTI	00011	xxxxxxxx	xxxxxxxx

ALU

Instruction	Op-Code	First Operand	Second Operand
NOT R_dst	01001	R_dst2 R_dst1 R_dst0	xxxxxxxx
ADD R_src, R_dst	01010	R_dst2 R_dst1 R_dst0	R_src2 R_src1 R_src0
SUB R_src, R_dst	01011	R_dst2 R_dst1 R_dst0	R_src2 R_src1 R_src0
AND R_src, R_dst	01100	R_dst2 R_dst1 R_dst0	R_src2 R_src1 R_src0
OR R_src, R_dst	01101	R_dst2 R_dst1 R_dst0	R_src2 R_src1 R_src0
INC R_dst	01110	R_dst2 R_dst1 R_dst0	xxxxxxxx
DEC R_dst	01111	R_dst2 R_dst1 R_dst0	xxxxxxxx
MOV R_src, R_dst	01000	R_dst2 R_dst1 R_dst0	R_src2 R_src1 R_src0

Schematic diagram of the processor

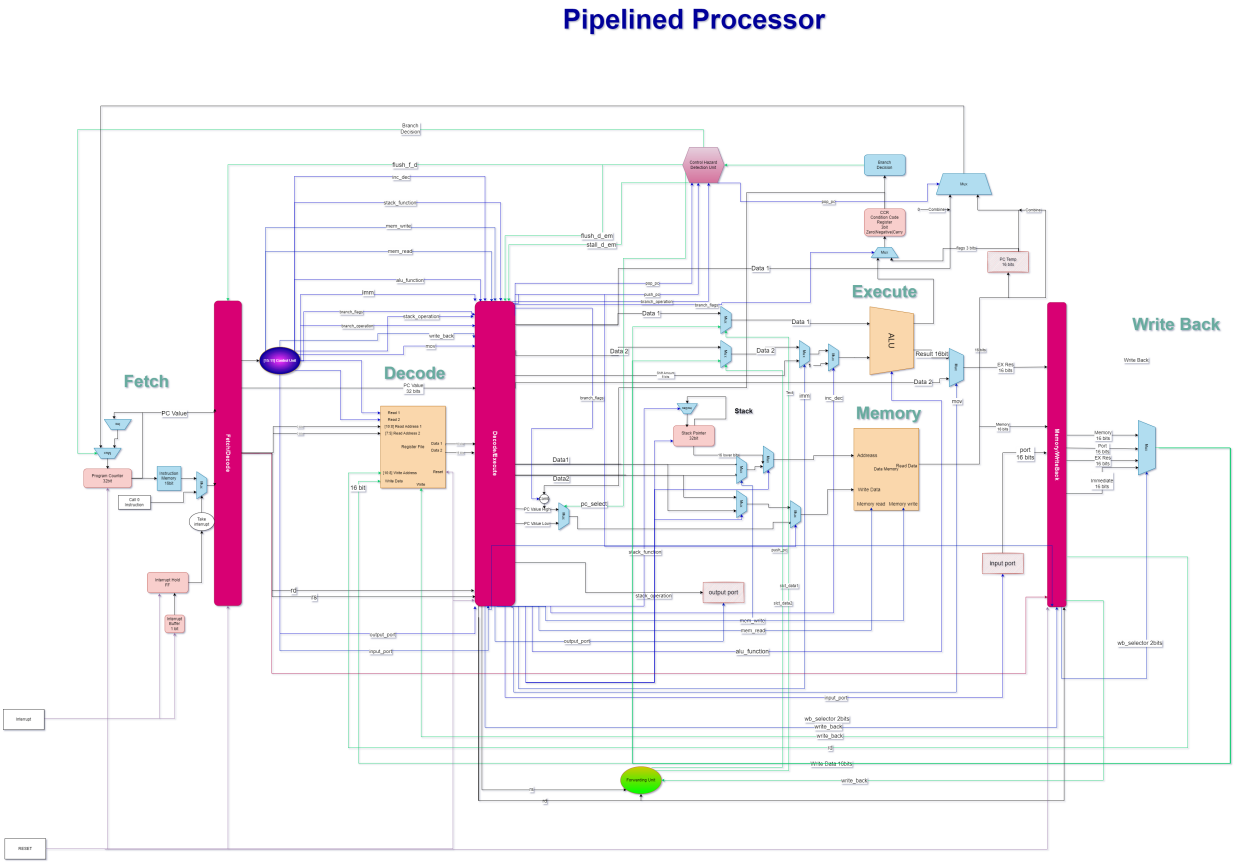

Figure 1: Design

Buffers

- 16-bit Fetch/Decode:

  - [15:0] Instruction

- 62-bit Decode/Execute-Memory:
  - input_port
  - output_port
  - mov
  - write_back
  - inc_dec
  - change_carry
  - carry_value
  - mem_read
  - mem_write
  - stack_operation
  - stack_function
  - branch_operation
  - imm
  - pop_pc
  - push_pc
  - branch_flags
  - [1:0] wb_selector
  - [2:0] alu_function
  - [2:0] branch_selector
  - [15:0] data1
  - [15:0] data2
  - [2:0] rd
  - [2:0] rs
- 70-bit Execute-Memory/WriteBack:
  - write_back
  - [1:0] wb_selector
  - [2:0] write_addr
  - [15:0] ex_result
  - [15:0] memory_data
  - [15:0] immediate
  - [15:0] port

Types of Hazards

1. Data-Hazards: Eliminated by Alu/Memory to Alu/Memory forwarding depending on the type of instruction.

   `Note: We don't have load use case since we are using only 4-stage processor. Both Alu/Memory Forwarding is from the result of the WriteBack mux.`

2. Control-Hazards:
   1. LDM: 2-memory locations instruction hence we need to fetch 2-times since DataBus is limited to 16-bit only. The second 16-bit fetched are data bits not instruction bits **thus we must flush the Decode/Execute-Memory buffer** in order to avoid executing the instructions' data bits.
   2. Call/Interrupt: Needs 2-cycles to push the 32-bit PC and the 3-bit flags.
   3. RTI/RET: Needs 2-cycles to fetch the 32-bit PC and the 3-bit flags from memory.

      `Note: PC is larger than memory address space so we use the most significant 3-bit to store the flags while push/pop.`

4. JZ/JN/JC/JMP: Static branch prediction with not taken. Value is ready in decode stage while the result is ready in execute stage.

Control Unit Design

wb_selector

- 00: pass execute
- 01: pass port
- 10: pass immediate
- 11: pass memory

Rtype: b15 = 0, **Itype**: b15 = 1, **Special**: b14 = 0

Rtype

ALU: b14 = 1

Function : b13-b11

For the following commands (b13-b11):

Not: 001 Add: 010 Sub: 011 And: 100 Or: 101

Control Signals:

- alu_function
  - alu_b13 = ~(inc/dec) & b13
  - (~b15 & b14) & alu_b13
  - (~b15 & b14) & b12
  - (~b15 & b14) & b11

    `Note: 000 is already no operation on alu. So there will be no issues when b13-b11 = 000`
- write_back: always 1
- wb_selector: 00

Also for the commands Inc: 110 Dec: 111 there are extra control signals:

- inc_dec: b13 & b12

Special

Special Command Mov : 000 with extra control signals: mov

For the following commands (b13-b11):

- NOP: 000
- SETC: 111
- CLRC: 110

Control Signals:

- Change_carry: b13 & b12
- carry_val: b11
- Call: 101

  - mem_write = 1
  - push_pc = 1
  - stack_function = 1
  - stack_operation = 1
  - branch_flags = interrupt
- Ret: 010
  - mem_read = 1
  - pop_pc = 1
  - stack_function = 1
  - stack_operation = 1
- Rti: 011
  - mem_read = 1
  - pop_pc = 1
  - stack_function = 1
  - stack_operation = 1
  - branch_flags = 1

Itype

For the following commands:

- Mem(b14-b13 = 00):
  - LDM(b12-b11 = 10): Imm, write_back, wb_selector: 10
  - LDD(b12-b11 = 11): MemRead, write_back, wb_selector: 11
  - STD(b12-b11 = 00): MemWrite
- Stack(b14-b13 = 01):
  - POP(b12-b11 = 11): MemRead, write_back, wb_selector: 11, stack_operation, stack_function must be 0
  - PUSH(b12-b11 = 00): MemWrite, stack_operation, stack_function = 1
- Branch(b14-b13 = 10):
  - JZ(b12-b11 = 00)
  - JN(b12-b11 = 01)
  - JC(b12-b11 = 10)
  - JMP(b12-b11 = 11)

    `All turn on the signal branch_operation, branch_selector = { b12, b11 }`
- Port(b14-b12 = 110):
  - IN(b11 = 0): write_back, wb_selector: 01
  - OUT(b11=1): output_port
- Alu(b14-b12 = 111):
  - SHL(b11=0)
  - SHR(b11=1)

    `All turn on: write_back, wb_selector: 00, imm, alu_function: {11, b11}`

Contributors

- Karim Mahmoud
- Mohamed Abdulhady
- Mohamed Kotb
- Mohamed Kamal