DLX

October 21, 2017

1 Introduction

The goal of this project is to implement a completely functional DLX processor in RTL VHDL, simulate it and then proceed with the design flow with synthesis and layout phases. DLX processor architecture was designed by John L. Hennessy and David A. Patterson and very well described and documented all over the internet. This project is developed for the course Microelectronic Systems taught by Prof. Mariagrazia Graziano in Politecnico di Torino.

Outline The remainder of this article is organized as follows. I will first give a brief overview of the DLX architecture and my personal implementation in Section 2. In Section 3 will go into details regarding particular aspects of the datapath and control unit. Section 3 and 6 are about the environment and results of design and layout phases. Finally, Section 4 gives the conclusions and possible future work.

2 Implmentation Overview

DLX is a classical 5-stage pipelined processor, with fixed length instructions and in-order execution. Most information regarding the ISA, structure and description of operations can be found in BLABLALBLALBLAA.

Instructions In this version, all I-TYPE and R-TYPE operations have been implemented with the exception of: LHI, RFE, TRAP, ITLB, LB, LH,

LBU, LHU, LF, LD, SB, SH, SF, SD and all MOV operations. The only F-TYPE instructions implemented are MULT and MULTU, but they operate on integer register instead of floating point ones.
During normal conditions, all operations take exactly 5 cycles to execute, with the exception of multiplications, which require a fixed amount of 13 cycles (9 cycles are spent in the execute stage) and they're not pipelineable.

Jump and branch Jump and Branch instructions target address is aggressively evaluated in the decode stage in order to have less cc penalty. In addition to this, a 2-bit predictor scheme has been implemented. This will be discussed more in the next section.

Control logic The execution is regulated by an hardwired CU, in charge of producing control words for each stage of the pipeline, as well as stall signals to stop the execution when needed. All possible hazards have been solved through a forwarding logic, when this is not enough, a stall is sent from the control unit.

Memories Two separated memories for instruction and data have been included in the design, but they're actually not part of it because they are always external peripherals.

- Instruction memory is asynchronous, with 1 read port. Write is not possible.
- Data memory is synchronous, with 1 read port and 1 write port. Concurrent write or read is not possible.

Register file Register file has 2 read port and one write port. Concurrent write and read on the same register produces a redirection of the input data to the output. Due to the position of the register file and the moment in which the control world is computed, it is not possible to selectively enable or disable read ports because it is not known a priori whether the instruction needs one or two source register.

3 Design Architecture

As previously mentioned, the processor is described in VHDL. Most blocks are written in a structural approach up to very basic logic elements such as multiplexers, half adders, etc ...
An overview of the processor architecture is shown in FIGUREXXX
In the next sections we are going to see in details each single component.

3.1 Fetch Block

Fetch block is in charge of correctly fetch the instruction to be executed from the memory. This block is tightly coupled with branch predictor. As can be seen in the picture, the actual decision of the next PC to drive comes from the BTB in case of normal operations.
MUXTARGET computes the correct target pc, which is sent to the branch predictor to evaluate the misprediction signal that controls MUXPRED. When misprediction signal is triggered, the current operation in the PC is not sent to decode stage, and it is replaced by the correct one on the successive clock cycle.

3.2 Branch Predictor

As previously stated, a branch predictor scheme have been applied, in particular it is a 2bit saturating counter BHT, also widely known in literature as (0,2). In addition, we have also implemented a BTB in charge of storing the PC of the next instruction in case the branch is taken.
The design applied is very generic, so the number of entries is easily configurable.
In normal situations the lowest part of the PC, called TAG, is used to index the BTH, if the instruction is recognized as a taken branch, then the PC found inside the BTB is sent to the fetch block. On the successive clock cycle, when the decode block actually computed the correct behavior of the branch, the value of the BTH is updated and, if the prediction was not correct, mispredict signal is triggered. Also, due to the fact that mispredict signal actually cause the current operation in the fetch stage to be re-evaluated, the branch predictor automatically disables itself on the following clock cycle to avoid deadlock conditions.

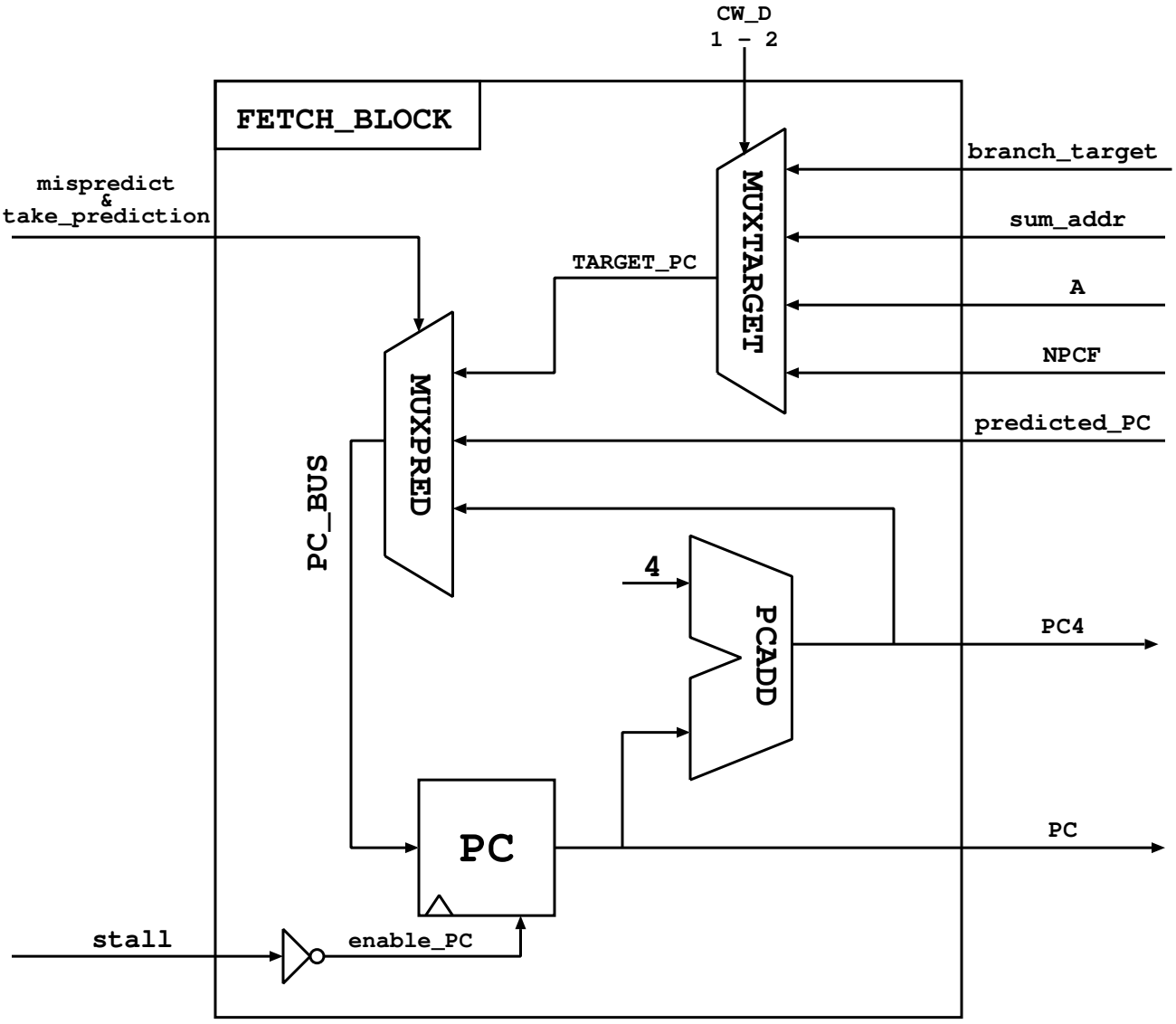

Figure 1: FETCHBLOCK

Some information regarding the operations performed on the last clock cycle need to be stored, to do so, 4 have been added, they store: prediction, prediction target, TAG and mispredict signals.

3.3 Decode Block

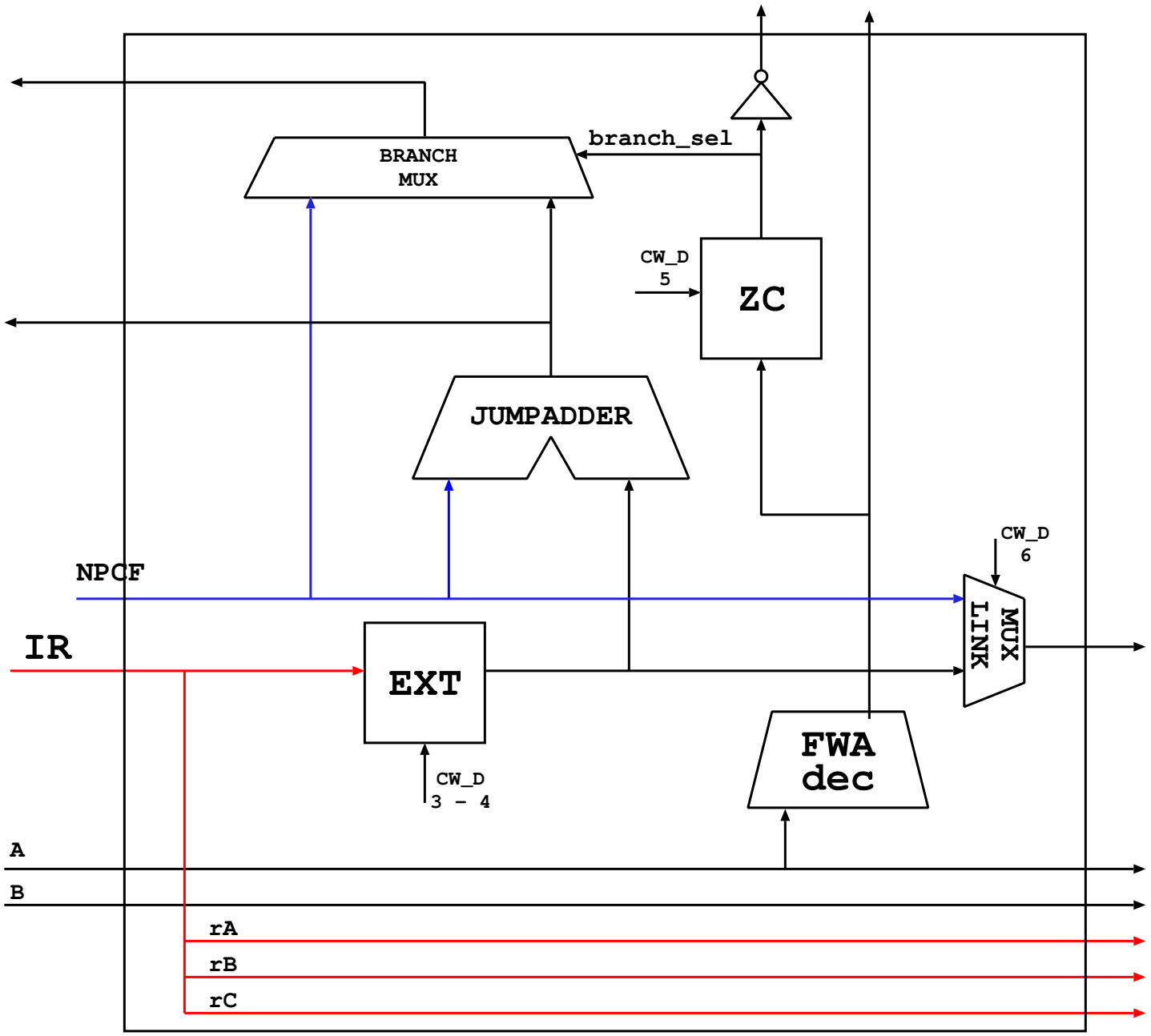

Figure 2: DECODEBLOCK

Decode block is in charge of 4 main functions:

- Split from IR register number values rA, rB and rC used in the next stage and for forwarding.
- Extend the immediate to 16/26 bits, either signed or unsigned, according to the current decoded instruction.
- Compute the addition between NPCF and the offset given by the extended immediate value
- Evaluate branch output through the zero comparator, the negation of this signal is also sent to other components in the system.

In case of Jump operations with link (JAL and JALR) the immediate value is replaced by the return address by MUXLINK.

3.4 Execute Block

BRIEF TALK ABOUT MUXES AT INPUT

3.5 ALU and Multiplier

ALU IN GENERAL COMPARATOR, SHIFTER, LOGIC UNIT, ADDER ALL OPERATIONS SHARE THE SAME ADDER. DESIGN FOR AREA REDUCTION

3.6 Memory Block

NOTHING MUCH TO SAY: ADDRESS ALWAYS COMES FROM ALU, DATA ALWAYS FROM S MUX TO IGNORE MEMORY NOT ENABLED WHEN NOT NEEDED

3.7 Forwarding Logic

FORWARDING CASES WHERE ARE FORWARDING MUXES PLACED

3.8 Control Unit

The control unit (CU) is in charge of generate and distribute control signal across the entire DLX pipeline. In this design, the control unit follows an Hardwired scheme, so at each IR input a kind of memory containing a control work for each instruction is looked up.
When a signal needs to be used later in the execution, it is propagated trough a series of flip flops.
The CU is able to correctly generate and handle stall conditions due to hazards or multi cycle operations. In general, when a stage is stalled, the same goes fol the preceding stages in the pipeline, while the successive need to continue the execution. When a stage would be empty, a BUBBLE operation is inserted: a BUBBLE has the same control word as a NOP. HARDWIRED INDEXING MOSTLY BEHAVIORAL STALL HANDLING WITH BUBBLES CONTROL WORD SIGNAL DESCRIPTION

S_MUX_PC_BUS - (1,2)

S_EXT - (3)

S_EXT_SIGN - (4)

S_EQ_NEQ - (5)

S_MUX_LINK - (6)

S_MUX_ALUIN - (7)

S_MUX_DEST - (8,9)

S_MUX_MEM - (12)

S_MEM_W_R - (11)

S_MEM_EN - (10)

S_RF_W - (13)

3.9 Stall Logic

LIST STALL CASES (HAZARDS) Control Hazards: MISPREDICTION Data hazards: RAW only: MEM/EXE solved by FW WB/EXE solved by FW DEC/EXE solved by stall: Branch or Jr situation
STALL COMING FROM EXECUTE Structural hazard: Multiplication multi cycle operation

3.10 Instruction and Data Memories

INSTRUCTION AND DATA MEMORIES NOT PART OF THE DESIGN ITSELF, SIZE OF DMEM AND IMEM As previously stated, Both memories are not of the design itself. Imem is accessed during fetch operation, no enable port because we need to fetch an operation at each clock cycle. Dmem accesses during MEM stage. There is an enable signal and a W/R. both read and write are synchronous during the falling edge of the clock. to move it at the rising edge need to move the control signal before, so new values are ready at the clock edge.

4 Simulation and Test

We worked hard, and achieved very little.

5 Synthesis

Not simple. separated CU and datapath. ST 65nm library. not suitable to synthesized memories, problems on BTB due to high number of lines. reduced to 8 lines for synthesis. all-flatten option gives extraordinary results but doesnt give informations on critical path.

6 Layout

In this section we describe the results.

References