



DLX project presentation

Dario Castagneri

Gianpietro Noto

Francesca Silvano

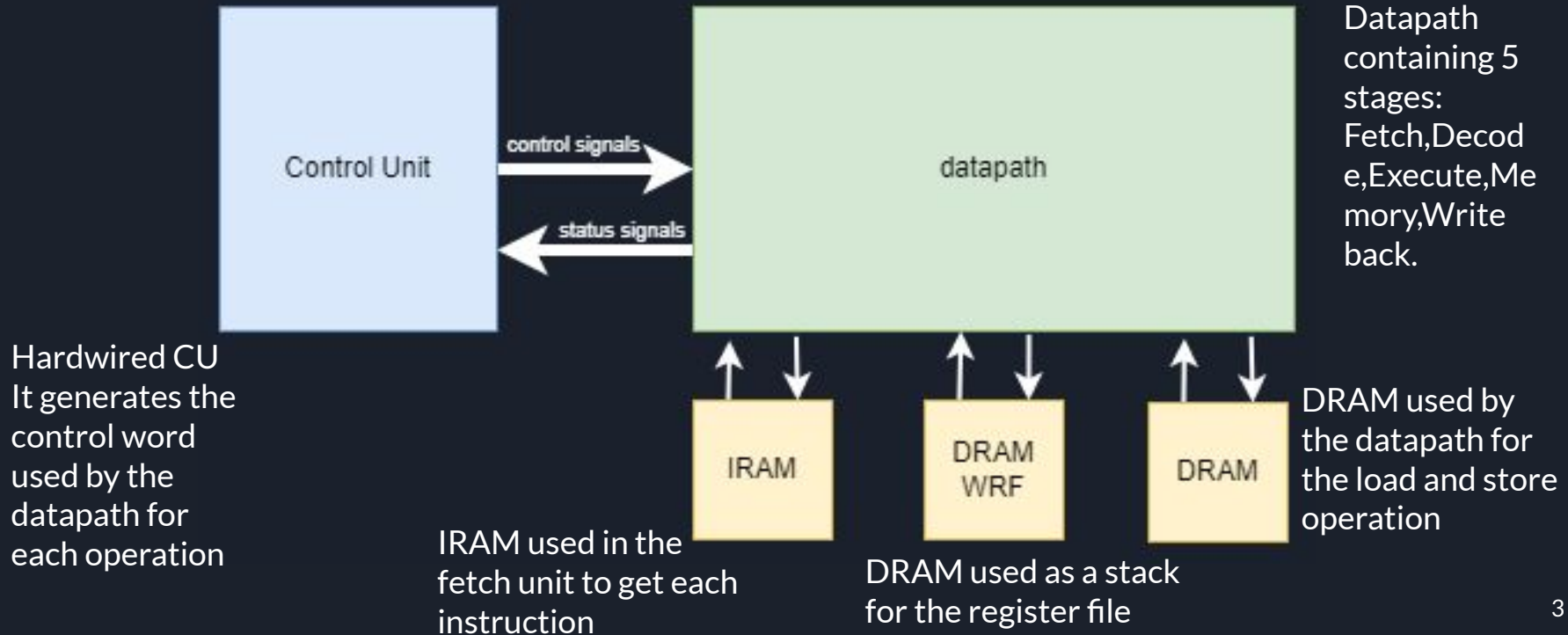
October 28th 2022



Outline

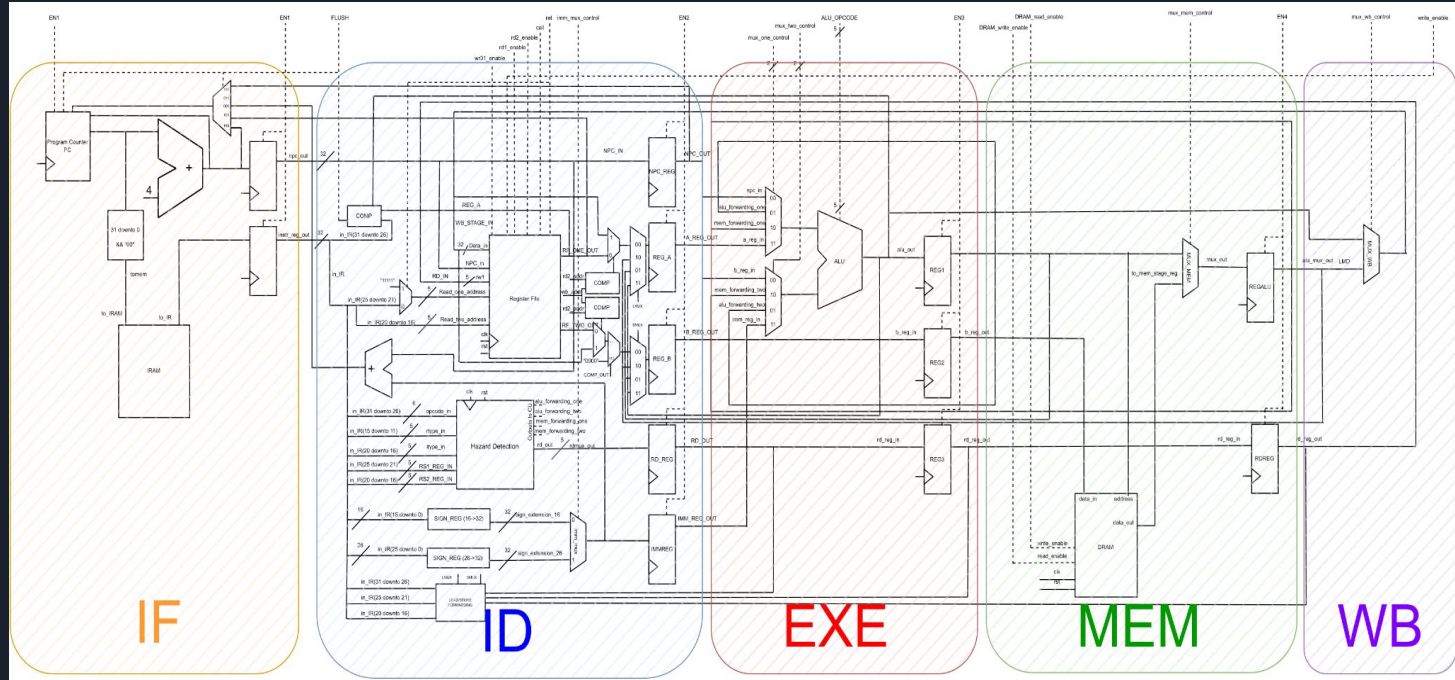
- Structural overview
- Datapath
- Hardwired Control Unit
- IRAM and DRAM
- Forwarding Load/Store
- Forwarding Hazards
- Windowed Register File
- Arithmetic-Logic Unit
- Automation scripts
- Synthesis Results
- Physical Design Results
- .asm file: load and store + jump instructions
- .asm file: forwarding hazards instructions
- .asm file: branch instructions

Structural Overview

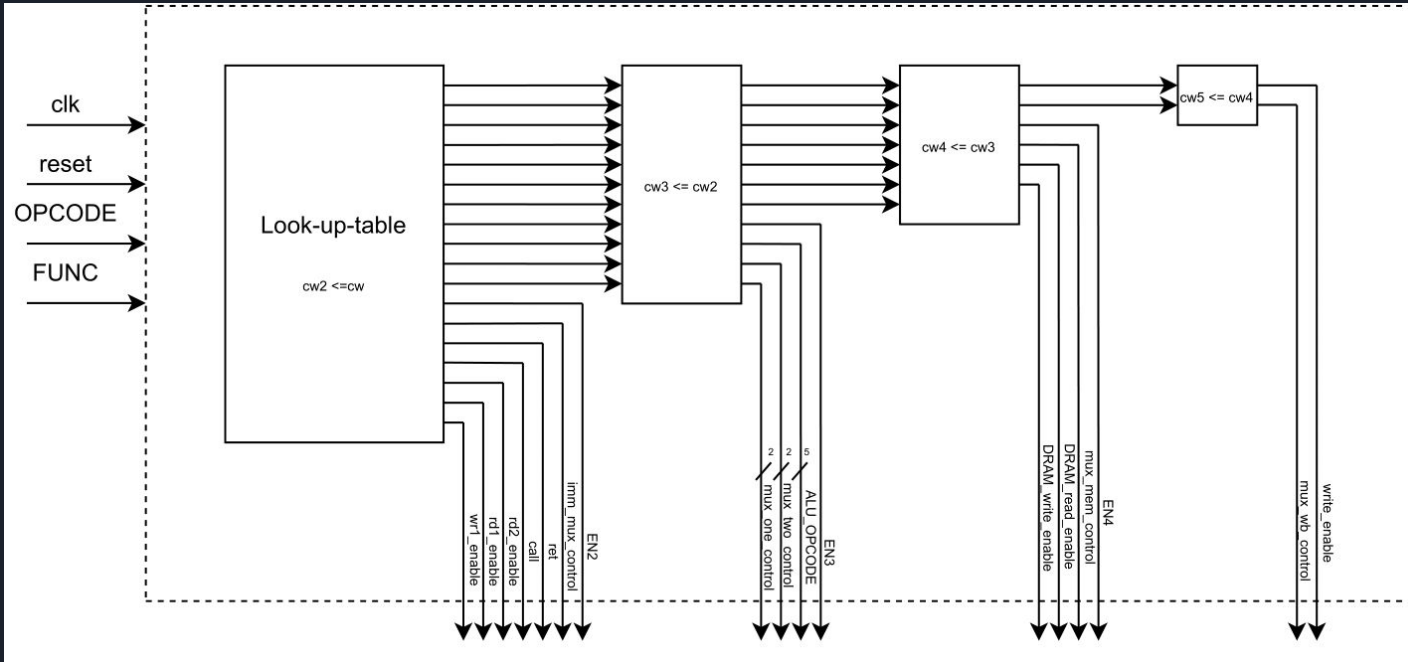


Datapath

The Datapath is divided in 5 stages. In this way the pipeline can be enabled and 5 instructions can be executed at a time.



Hardwired Control Unit



Hardwired CU, it generates the control words used by the datapath to execute instructions.

When a new instruction is fetched by the datapath, the OPCODE and the FUNC are sent to the CU which generates the Control Word.

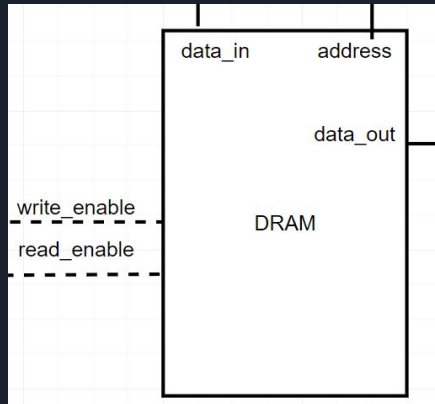
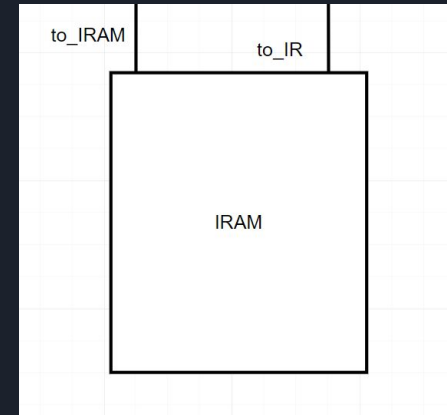
On the following clock cycle the generated Control Word is sent to the datapath which can perform the Decode of the fetched instruction.

IRAM and DRAM

Cells are made of 8-bit, instructions are in 32-bit so they need 4 cells to be stored; memory organization is in Big Endian.

When the Reset = 1 the IRAM reads the file containing the instructions and store them.

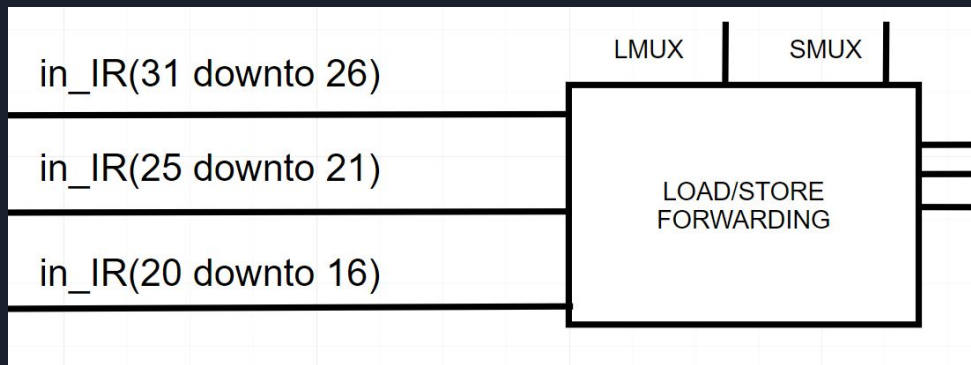
Then when an external enable is activated every instruction is fetched in the rising clock edge and is passed to the following stage through an Instruction Register IR.



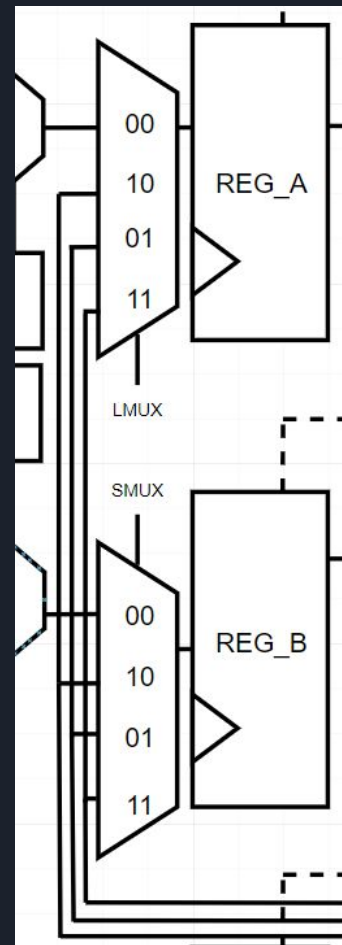
The DRAM is used in presence of Load/Store instructions. In order to read or store instruction the respective enable signals must be activated.

Since we are in a pipeline both memories must work in a combinatory way so that when the right address/data_in and write/read signals are given, the correct operation is performed.

Forwarding Load/Store



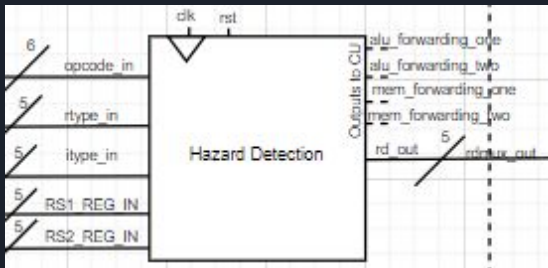
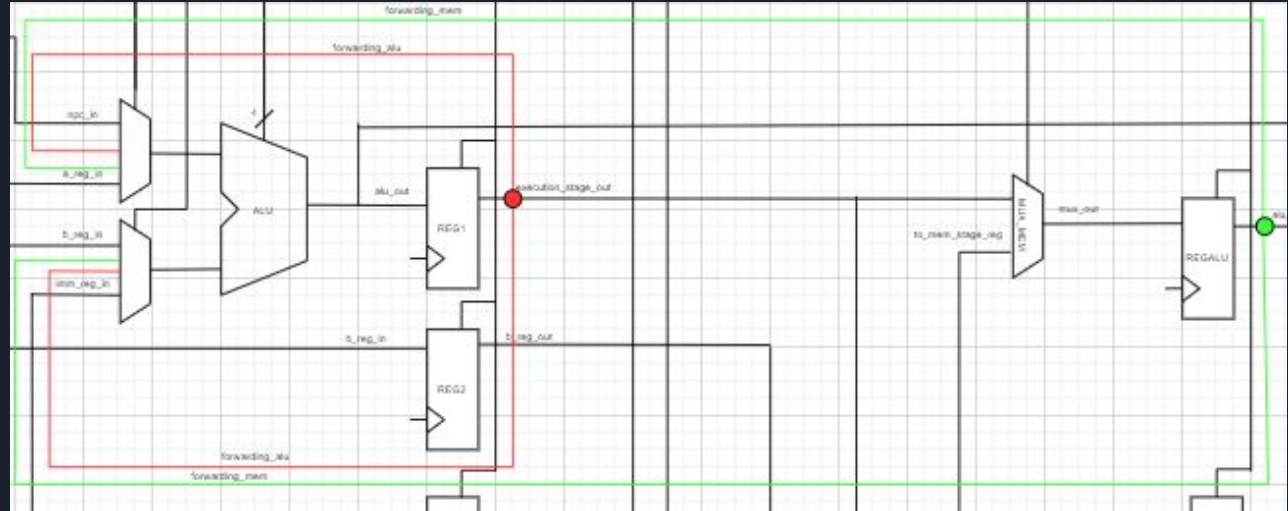
The Load/Store Forwarding block is used to enable forwarding in presence of a Data Hazard for Load or Store instructions. The inputs of this block are the Opcode, the read 1 address (RS1) and the read 2 address (RS2) of the Register File, while on the other side we have in input the Destination registers RD from the following stage of the Pipeline. If the data that in the following clock cycles must be written in the Register File is present in the Pipeline, it is forwarded directly to the output of the Decode Unit; the right value is selected through the two 4TO1 Multiplexers used respectively for Load and Store instructions.



Forwarding Hazards

The DLX contains inside the decode phase a block to manage links between register of close instructions.

4 signal for forwarding

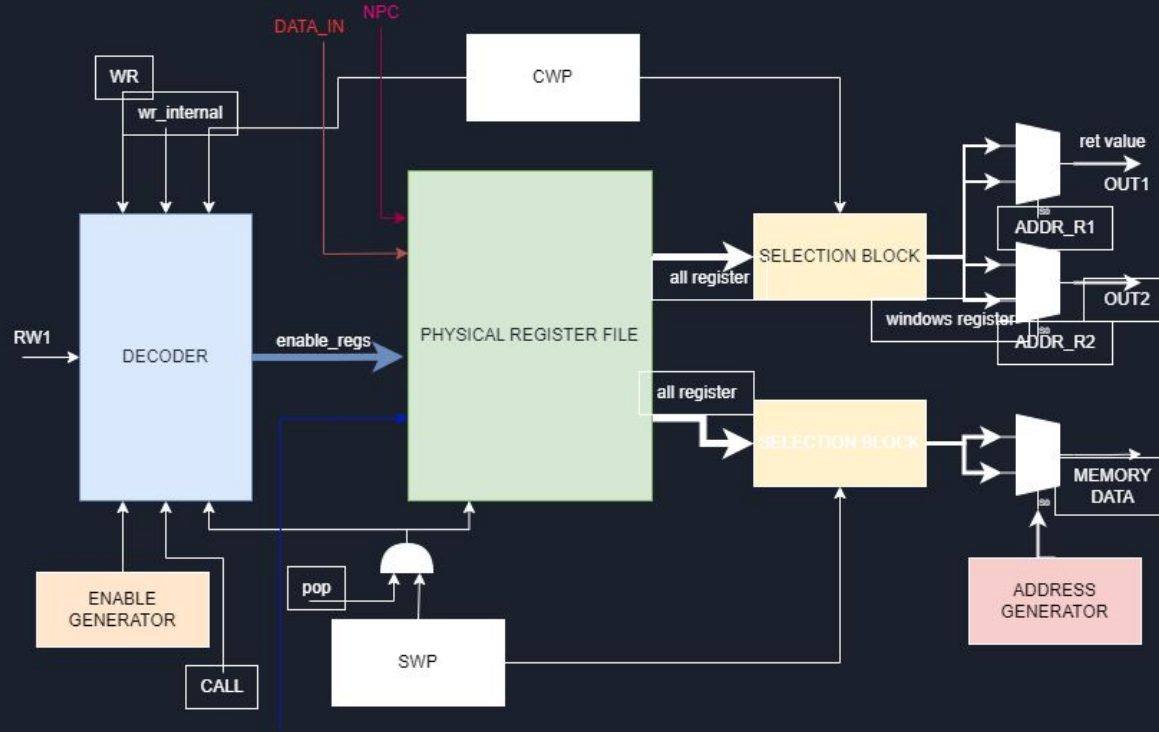


1. It check the opcode of the previous instruction with the new
2. if some hazard is possible checks the register destination with the register source
3. If the forwarding can be managed it pull up one of the signals otherwise the signal nop_add is pulled up

Windowed Register File

call signals
and ret
signals
perform the
switch
context

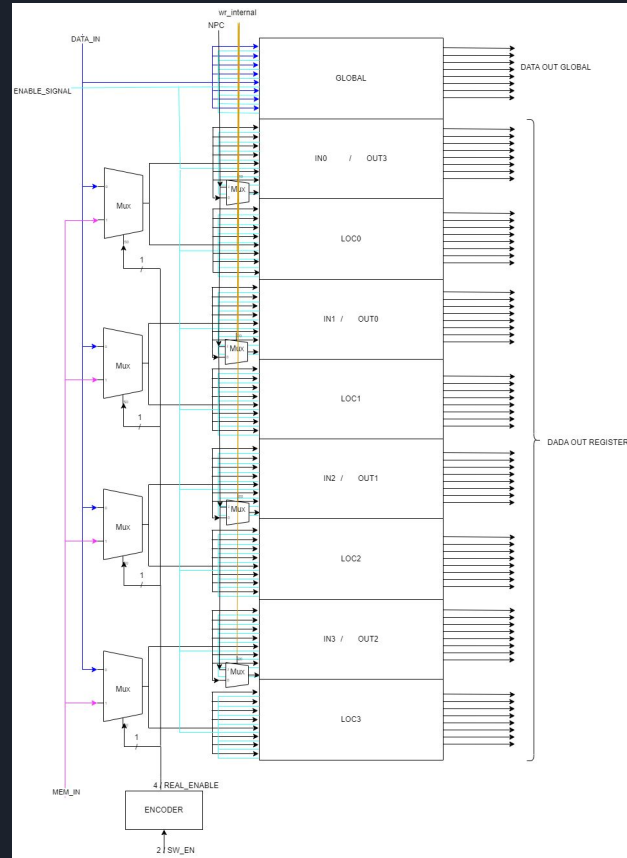
enable
generators
is used for
the pop
from the
stack



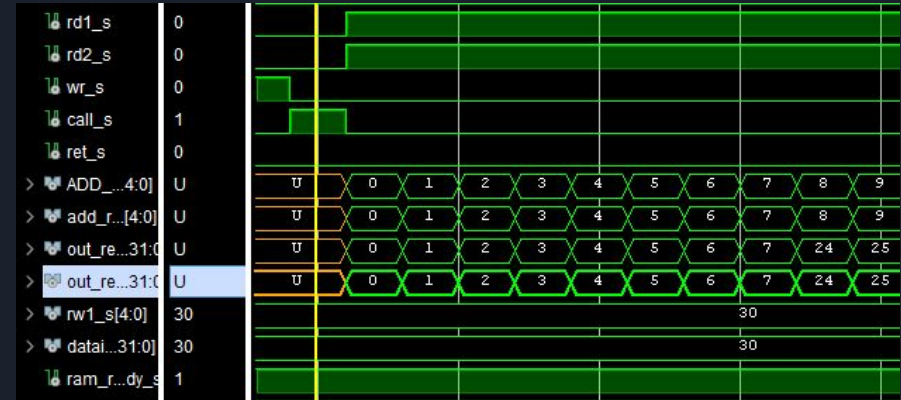
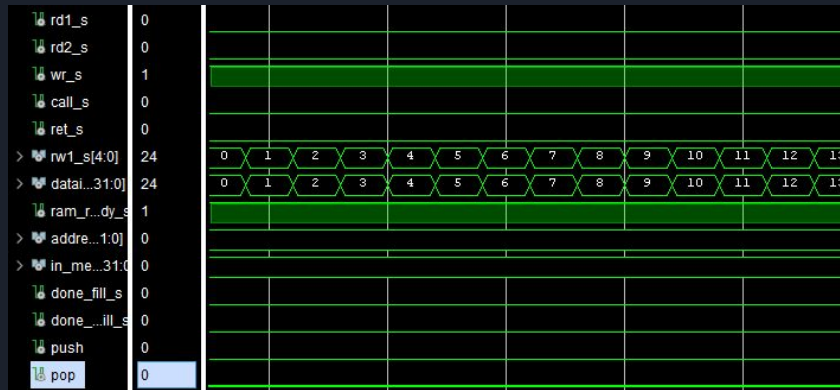
add_r1 and
add_r2
selects the
register going
in the exit

The address
generator is
useful when a
push in the
stack is
performed

Windowed register file Physical register file



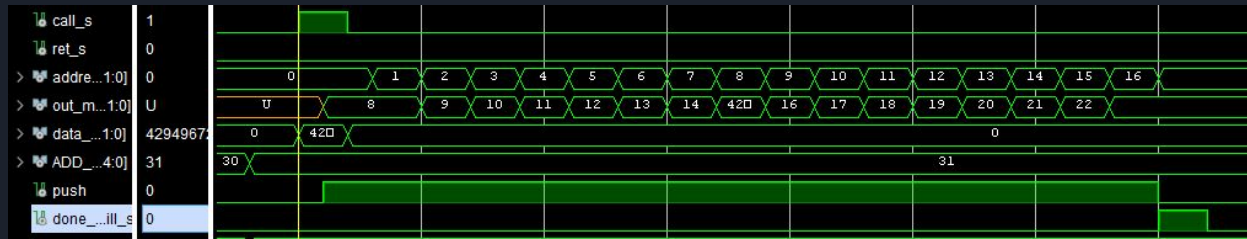
Windowed register file write and read operation test



The signal for the write `wr_s` is up, so the wrf take the data and the address from the outside and write in the correct register of the current windows.

Here a call is performed and a read of the input register from the previous windows.

Windowed register file call and return operation test

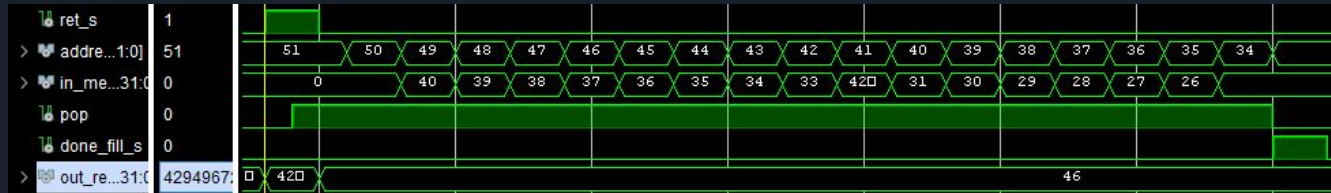


The call operation with the use of the stack is quite easy:

- the address to return is saved in the R31

- the load of the registers is done when the push signal is at one

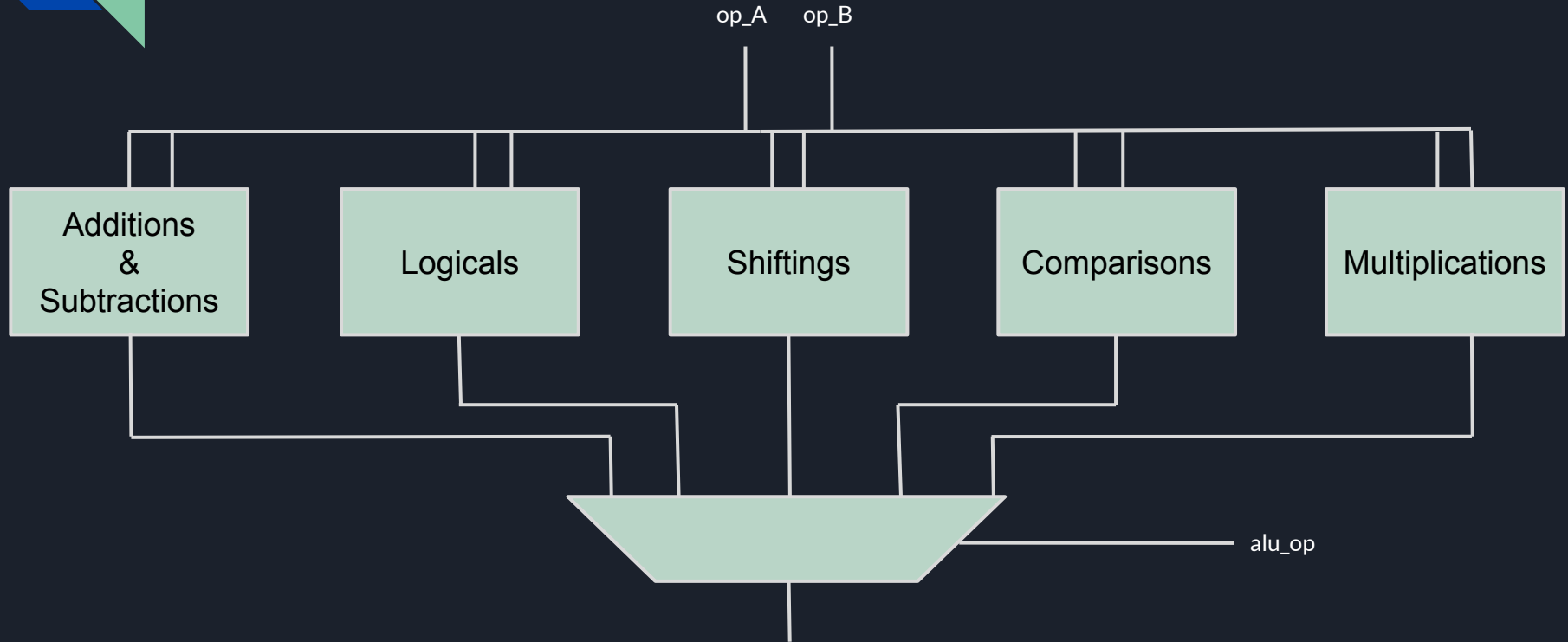
- when all is finished the done_fill signal is up



When a Return operation is performed:

- the address of the instruction is given in output in the first clk cycle

Arithmetic-Logic Unit






Automation scripts

- simulation.tcl
 - To verify the dlx correctness using custom assembly files
 - compile_directory
 - simulate_dlx
- synthesis.tcl
 - To perform the dlx synthesis and extract reports
 - analyze_directory
 - synthesize (single threshold voltage with NandgateOpenCellLibrary 45 nm)
 - synthesize_dual_vth (double threshold voltage with STcmos 65 nm)



Synthesis results

Synthesis	Library	Clock period	Total Dynamic Power	Total area
Standard	NandGateOpenCellLibrary	2 ns	6.79 mW	1584 μm^2
Dual V-th	STcmos65	2 ns	6.75 mW	3625 μm^2



HVT 5.74 %
LVT 94.26 %



Physical design results

Power report		
Internal	4.959 mW	57.61%
Switching	3.465 mW	40.25%
Leakage	0.1832 mW	2.12%
Total	8.608 mW	

Area report	
Total area	8961.3 μm^2

Other available reports:

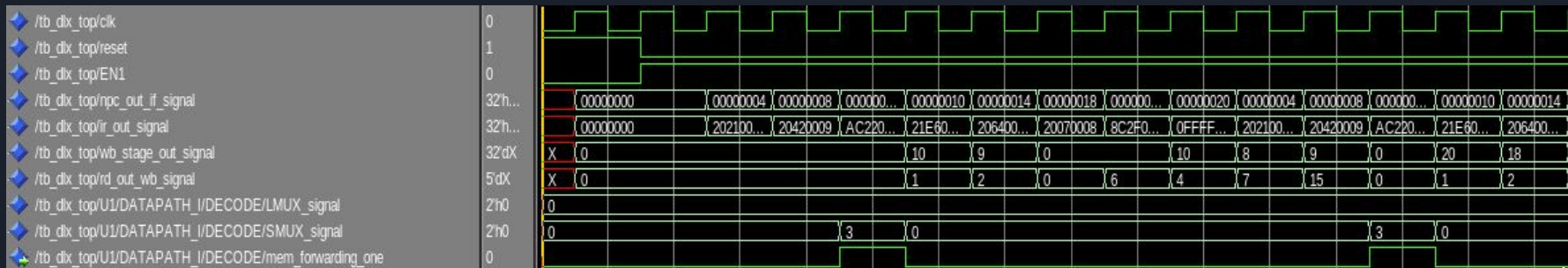
- setup and hold timing
 - postCTS
 - postRoute
- parasitics
 - resistances
 - capacitances

.asm file: load/store + jump instructions

```

1  myloop:
2
3  addi r1, r1, #10 ;   r1 <= 10
4  addi r2, r2, #9  ;   r2 <= 9
5  ✓ sw 10(r1), r2    ;   r1 mem_forwarding_one and r2 forwarding
6  | | | | | ;   Address = 20   Data = 9
7  addi r6, r15, #0 ;   r6 <= 0
8  addi r4, r3, #10 ;   r4 <= 10
9  addi r7, r0, #8  ;   r7 <= 8
10 lw r15, 10(r1)    ;   Address = 20   Data = 9
11 jal myloop        ;   Jump and link
    
```

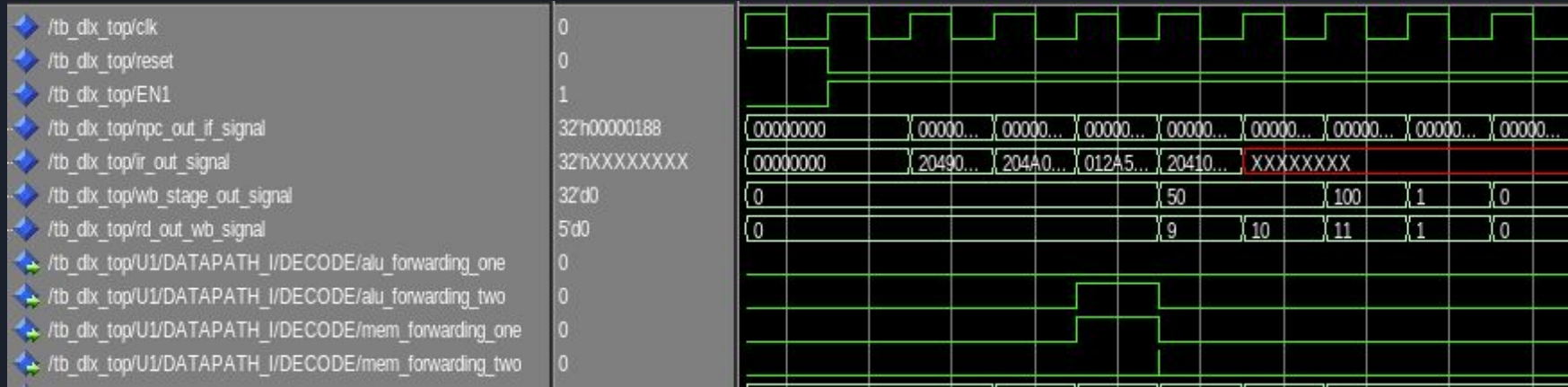
When the store is decoded the mem_forwarding_one for r1 is enabled and the forwarding for r2 is activated.



.asm file: forwarding hazards instructions

We expect signals `alu_two_forwarding` and `mem_one_forwarding` active in the third instruction.

```
1 addi r9, r2, #50 ; r9 <= 50
2 addi r10, r2, #50 ; r10 <= 50
3 add r11, r9, r10 ; r11 <= 100
4 addi r1, r2, #1 ; r1 <= 1
```



.asm file: branch instructions

```
1  addi r1, r0, 100
2  xor r2, r2, r2
3
4  ciclo:
5  addi r3, r3, 10
6  subi r1, r1, 1
7  addi r2, r2, 4
8  bnez r1, ciclo
9
10 addi r4, r0, 65535
11 ori r5, r4, 10000
```

The value of r1 is initialized in the first instruction, then it's decreased by the sub instruction and the value of r1 is compared by the bnez instruction. When the bnez is in decode it requires the content of the register r1 to perform the comparison, so r1 is forwarded directly at the input of the alu in the EXE stage. We expect the signal mem_forwarding_one active in the decode of the bnez instruction.

