[CS-523] Digital Circuits Design Lab using EDA Tools
Computer Science Department
University of Crete

Implementation of 32-bit RISC-V, in SystemVerilog using Synopsys' EDA tools

Team:

1. Ioannis Vardas (vardas@csd.uoc.gr)
2. Antonios Psistakis (psistakis@csd.uoc.gr)

5 July 2018

Roadmap

- Introduction
- Stage 1: Instruction Fetch
- Stage 2: Instruction Decode
- Stage 3: Execution
- Stage 4-5: Data Memory, Write Back
- Evaluation
- Future Work/Conclusion

Roadmap

- Introduction
- Stage 1: Instruction Fetch
- Stage 2: Instruction Decode
- Stage 3: Execution
- Stage 4-5: Data Memory, Write Back
- Evaluation
- Future Work/Conclusion

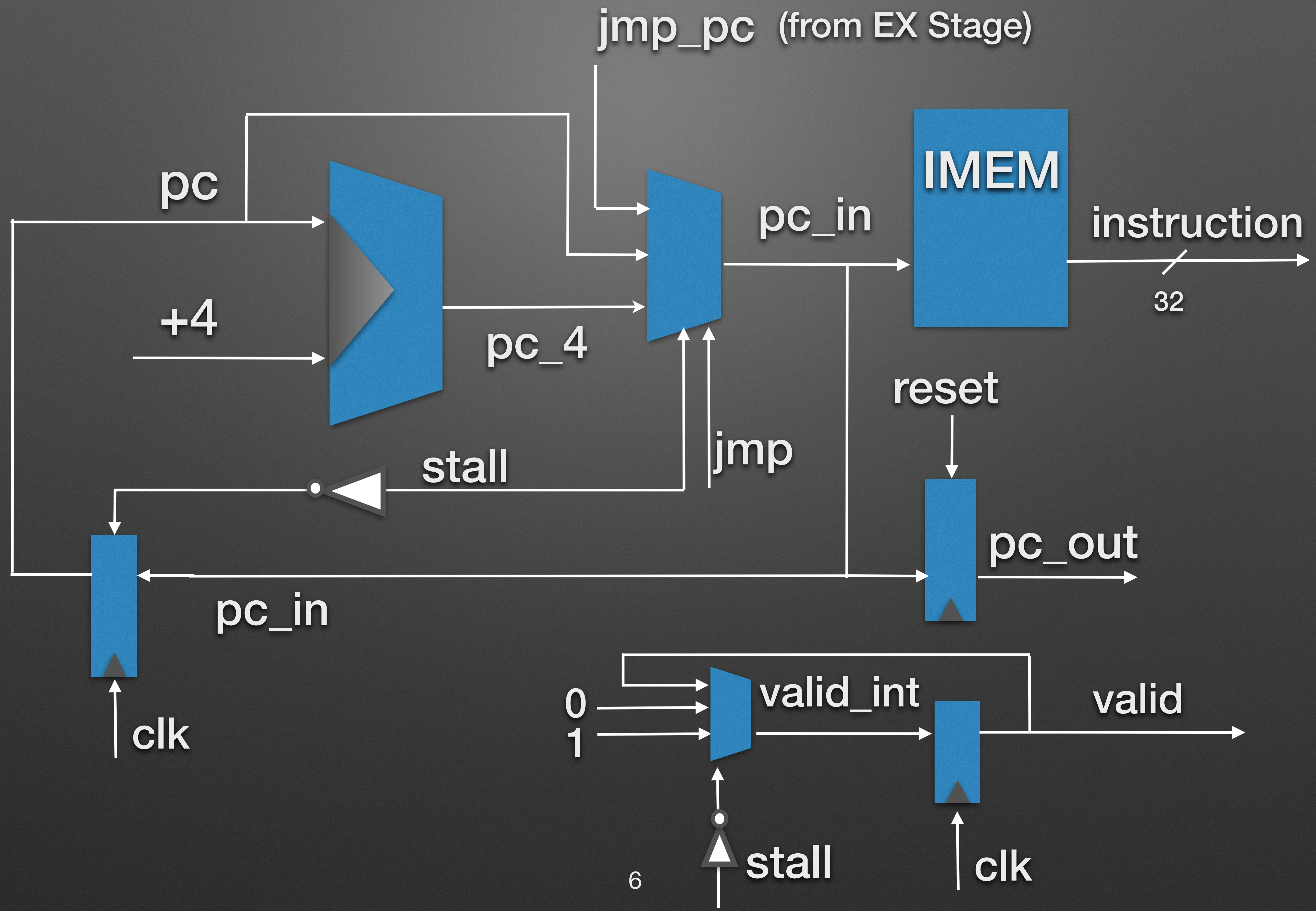
Introduction

- RISC-V is an open instruction set architecture (ISA) based on established reduced instruction set computing (RISC) principles
    - Secrecy prevents public educational use, security auditing, and development of public, low-cost free and open-source software compilers, and operating systems
- Not the first open architecture ISA, but significant → designed to be useful in modern computerized devices such as warehouse-scale cloud computers, high-end mobile phones and the smallest embedded systems

Roadmap

- Introduction
- Stage 1: Instruction Fetch
- Stage 2: Instruction Decode
- Stage 3: Execution
- Stage 4-5: Data Memory, Write Back
- Evaluation
- Future Work/Conclusion

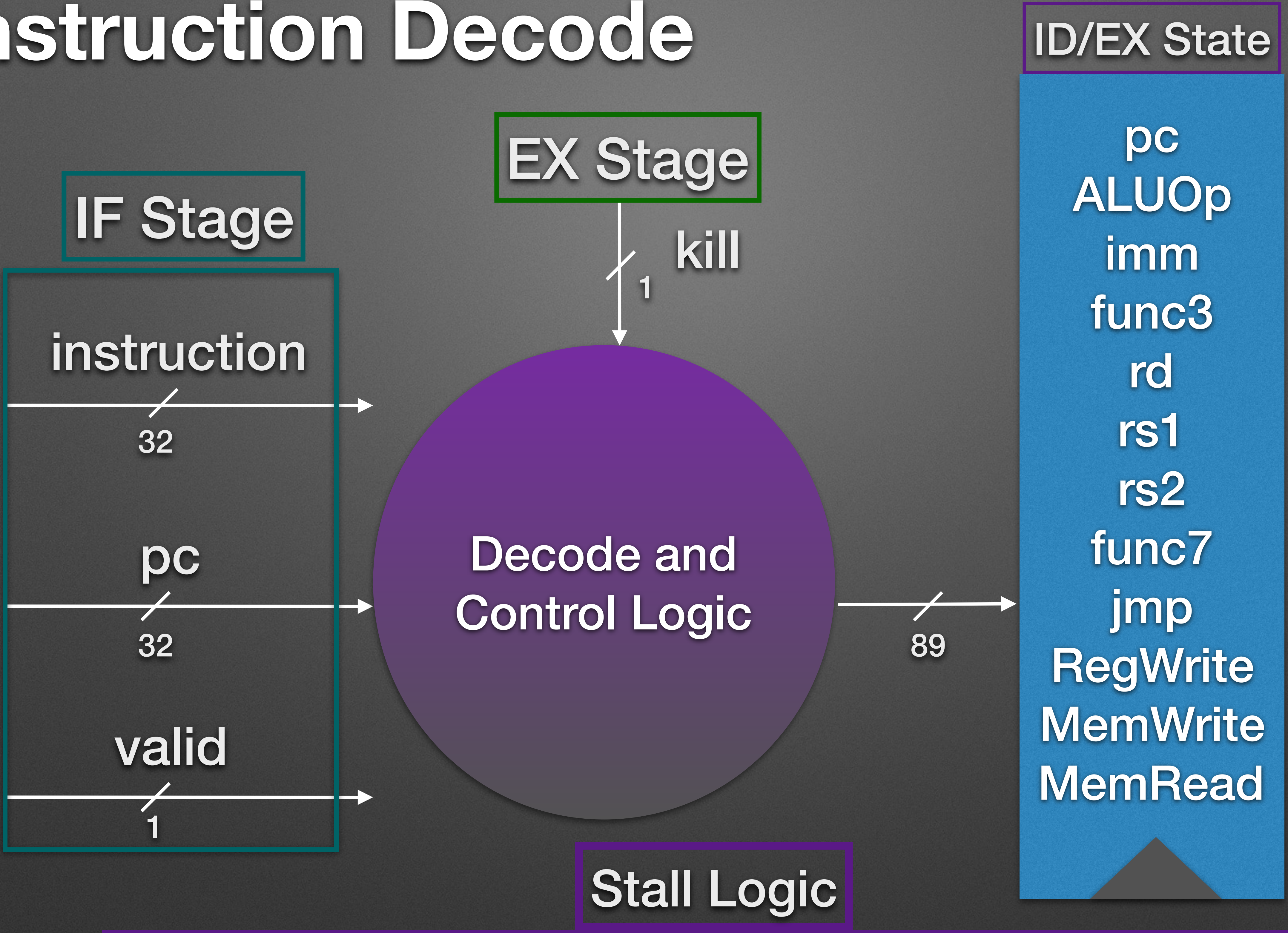
Instruction Fetch



Roadmap

- Introduction
- Stage 1: Instruction Fetch
- Stage 2: Instruction Decode
- Stage 3: Execution
- Stage 4-5: Data Memory, Write Back
- Evaluation
- Future Work/Conclusion

Instruction Decode



```
if( ((instruction[24:20] == ID_EX_state.rd || instruction[19:15] == ID_EX_state.rd)
        && ID_EX_state.rd != 5'b0 && ID_EX_state.RegWrite == 1 ) ||
    ((instruction[24:20] == MEM_state.rd || instruction[19:15] == MEM_state.rd )
        && MEM_state.rd != 5'b0 && MEM_state.RegWrite == 1) )
```

Roadmap

- Introduction
- Stage 1: Instruction Fetch
- Stage 2: Instruction Decode
- Stage 3: Execution
- Stage 4-5: Data Memory, Write Back
- Evaluation
- Future Work/Conclusion

Execution

- Used a new implementation that requires 3 bits generated by Decode stage that will drive the whole execution (and the ALU)

different unit for jmp

ALUop	Type	Operation
000	LD, ST	ALUout = i_A + i_Imm_SignExt

001	BRANCH	jmp_PC = i_NPC + i_Imm_SignExt
		jmp = 1 (flag)

010	R-TYPE	based on requested op
011	I-TYPE	based on requested op
100	LUI	ALUout = i_Imm_SignExt
101	AUIPC	ALUout = i_NPC + i_Imm_SignExt

110	JAL	jmp_PC = i_NPC + i_Imm_SignExt
		jmp = 1 (flag)

111	JALR	jmp_PC = i_A + i_Imm_SignExt
		jmp = 1 (flag)

Execution

- Used a new implementation that requires 3 bits generated by Decode stage that will drive the whole execution (and the ALU)

different unit for jmp

ALUop	Type	Instructions	Operation
000	LD, ST	LB, LH, LW, LBU, LHU, SB, SH, SW	ALUout = i_A + i_Imm_SignExt

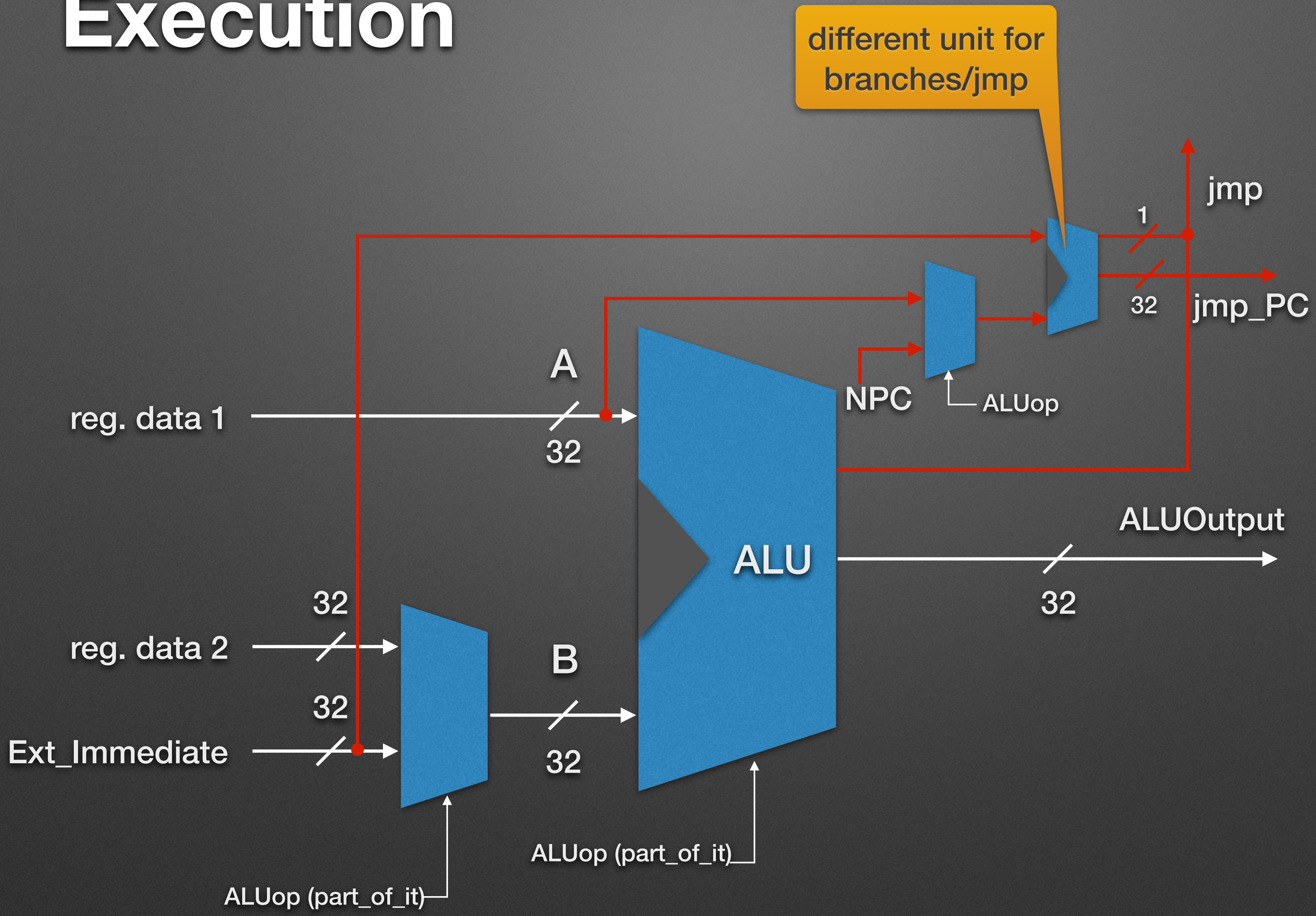
001	BRANCH	BEQ, BNE, BLT, BGE, BLTU, BGEU	jmp_PC = i_NPC + i_Imm_SignExt
			jmp = 1 (flag)

010	R-TYPE	ADD, SUB, SLL, SLT, SLTU, XOR, SRL, SRA, OR, AND	based on requested op
011	I-TYPE	ADDI, SLTI, SLTIU, XORI, ORI, ANDI, SLLI, SRLI, SRAI	based on requested op
100	LUI	LUI	ALUout = i_Imm_SignExt
101	AUIPC	AUILPC	ALUout = i_NPC + i_Imm_SignExt

110	JAL	JAL	jmp_PC = i_NPC + i_Imm_SignExt
			jmp = 1 (flag)
			o_ALUOutput = i_NPC + 4

111	JALR	JALR	jmp_PC = i_A + i_Imm_SignExt
			jmp = 1 (flag)
			o_ALUOutput = i_NPC + 4

Execution


different unit for branches/jmp
jmp
1
32
jmp_PC
A
NPC
ALUop
reg. data 1
32
ALU
ALUOutput
32
32
reg. data 2
32
B
Ext_Immediate
32
ALUop (part_of_it)
ALUop (part_of_it)

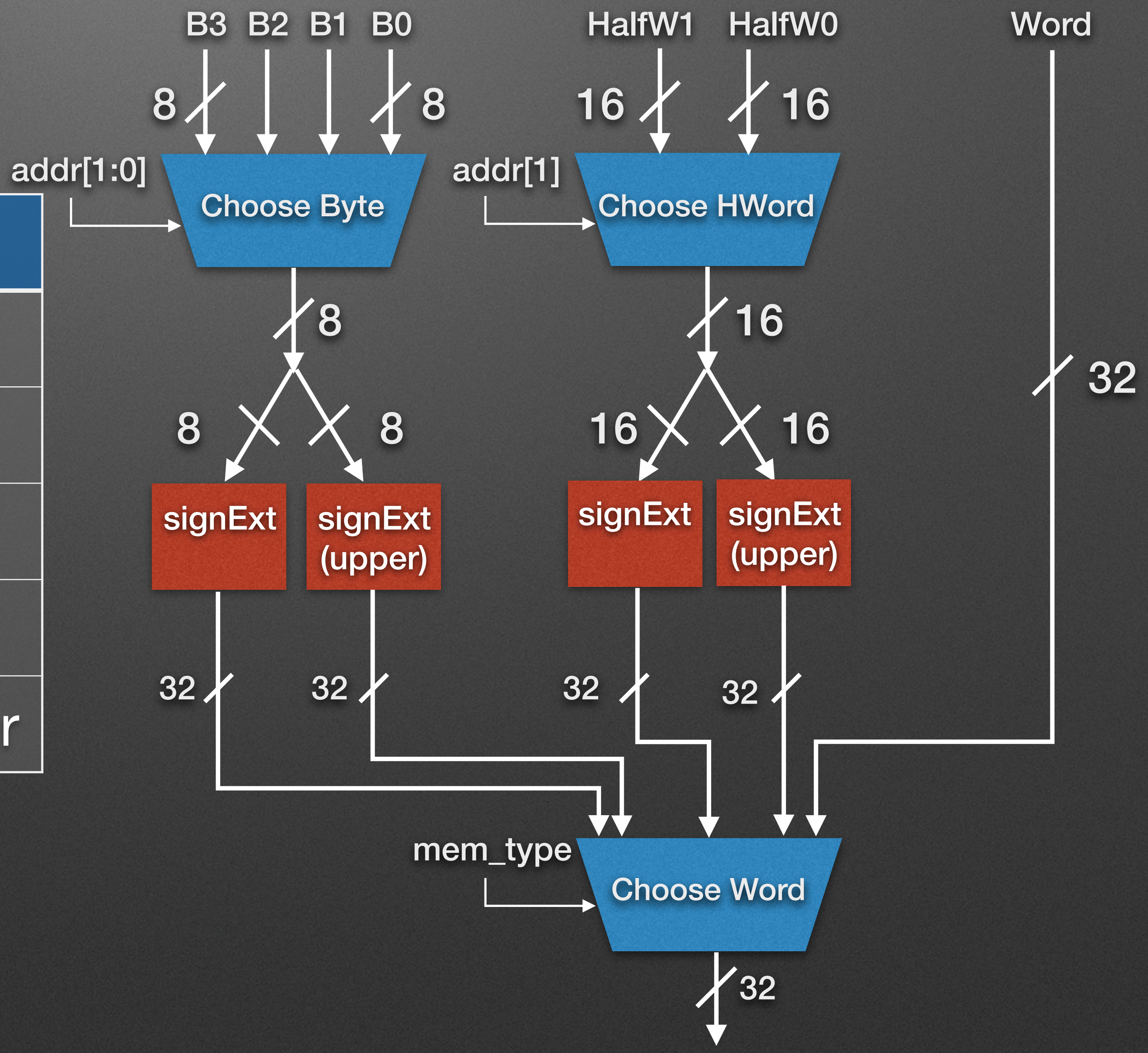
Roadmap

- Introduction
- Stage 1: Instruction Fetch
- Stage 2: Instruction Decode
- Stage 3: Execution
- Stage 4-5: Data Memory, Write Back
- Evaluation
- Future Work/Conclusion

Data Memory & Write-Back

- Loading from memory → Use of some mux'es…

Instr.	Meaning
LB	load byte
LH	load half-word
LW	load word
LBU	load byte upper
LHU	load half-word upper



Roadmap

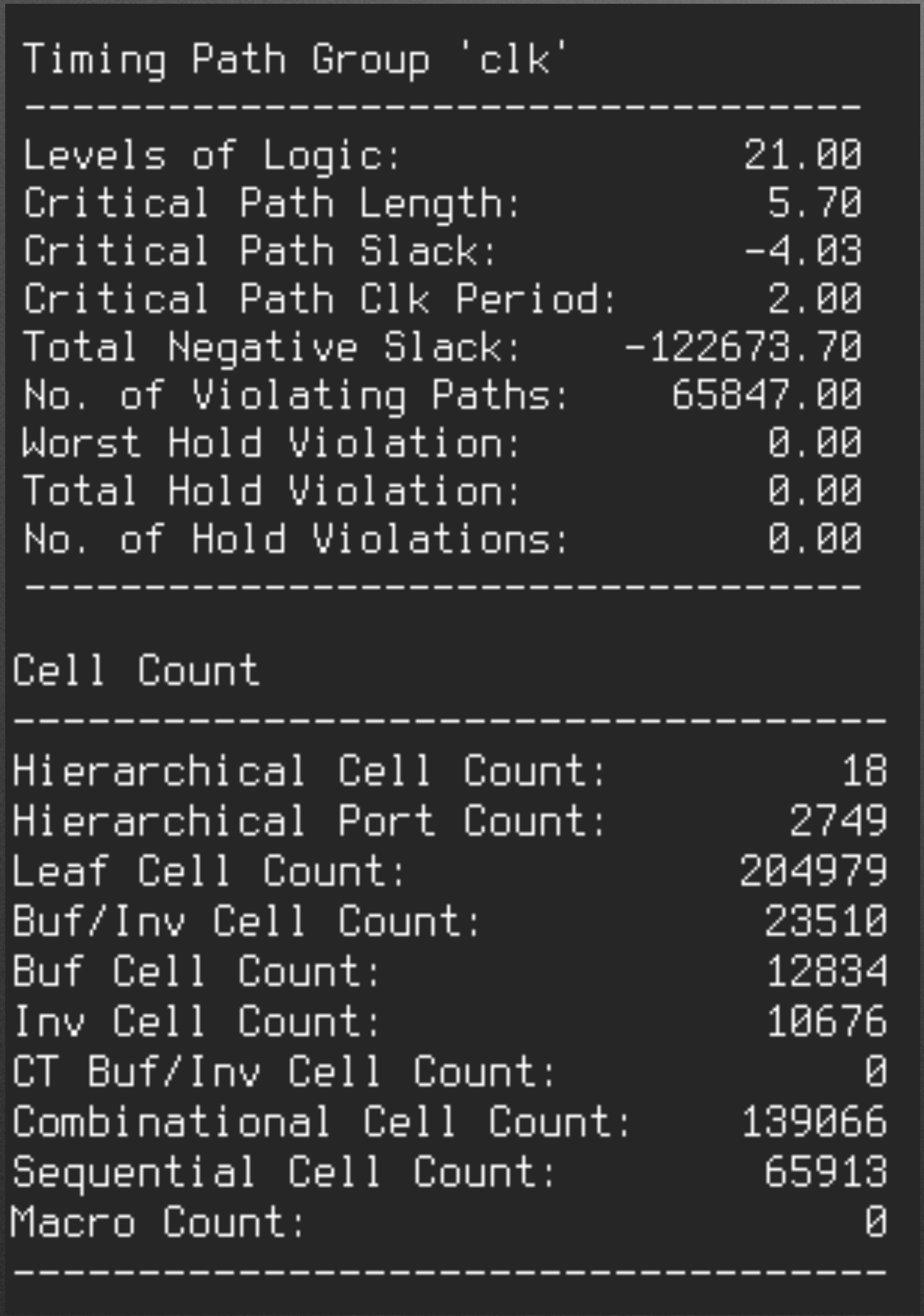
- Introduction
- Stage 1: Instruction Fetch
- Stage 2: Instruction Decode
- Stage 3: Execution
- Stage 4-5: Data Memory, Write Back
- Evaluation
- Future Work/Conclusion

Evaluation

- **Tests we ran**
  - Given examples (Example 0, 1, 2)
  - Fibonacci (n=16)
  - All types of load and store instructions
  - I-type instructions
  - And more…
- **Synthesis with**
  - 500 MHz (violated)
  - 150 MHz

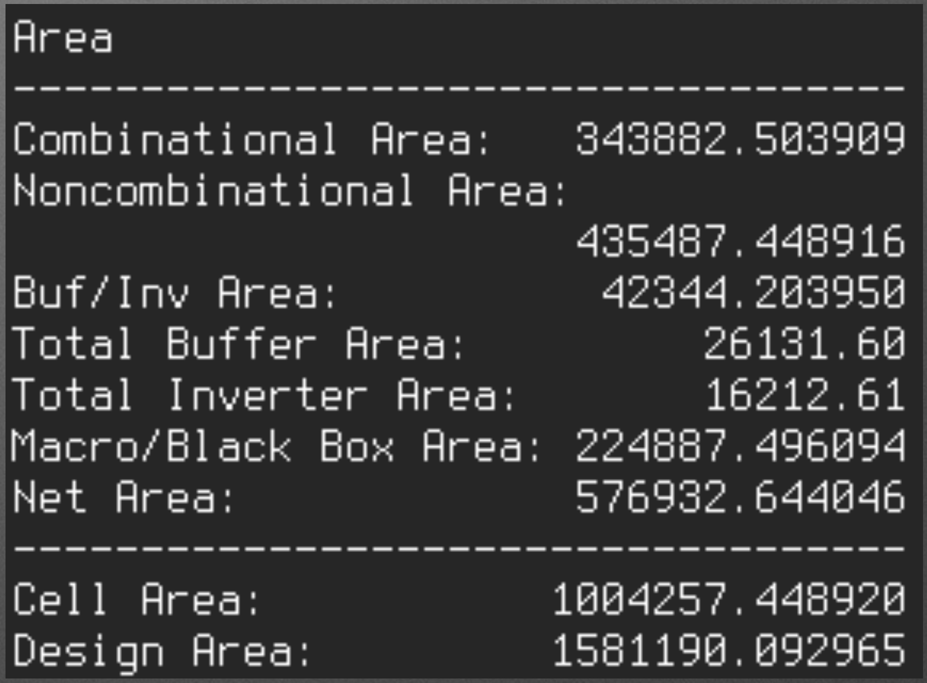
Evaluation

- **500 MHz clk**

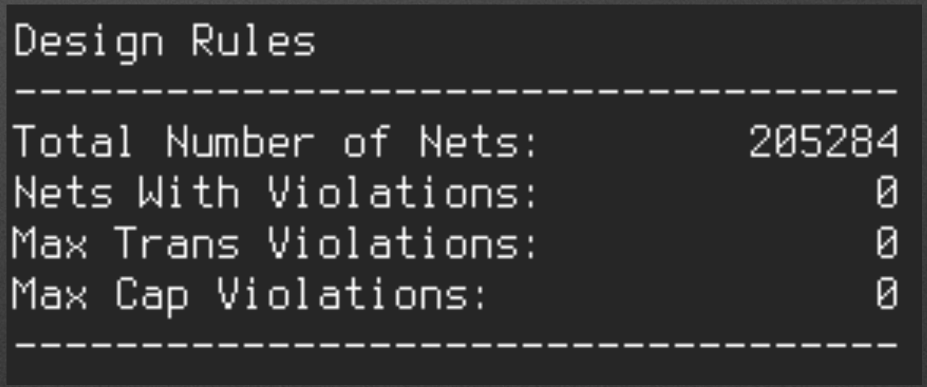


```
Timing Path Group 'clk'
-----------------------------------
Levels of Logic:               21.00
Critical Path Length:           5.70
Critical Path Slack:           -4.03
Critical Path Clk Period:       2.00
Total Negative Slack:     -122673.70
No. of Violating Paths:     65847.00
Worst Hold Violation:           0.00
Total Hold Violation:           0.00
No. of Hold Violations:         0.00
-----------------------------------

Cell Count
-----------------------------------
Hierarchical Cell Count:          18
Hierarchical Port Count:        2749
Leaf Cell Count:              204979
Buf/Inv Cell Count:            23510
Buf Cell Count:                12834
Inv Cell Count:                10676
CT Buf/Inv Cell Count:             0
Combinational Cell Count:     139066
Sequential Cell Count:         65913
Macro Count:                       0
-----------------------------------
```



```
Area
-----------------------------------
Combinational Area:   343882.503909
Noncombinational Area:
                      435487.448916
Buf/Inv Area:          42344.203950
Total Buffer Area:         26131.60
Total Inverter Area:       16212.61
Macro/Black Box Area: 224887.496094
Net Area:             576932.644046
-----------------------------------
Cell Area:           1004257.448920
Design Area:         1581190.092965
```



```
Design Rules
-----------------------------------
Total Number of Nets:         205284
Nets With Violations:              0
Max Trans Violations:              0
Max Cap Violations:                0
-----------------------------------
```

Power

Global Operating Voltage = 1.05
Total Dynamic Power = 356.4417 mW (100%)
Cell Leakage Power = 89.8461 mW

Roadmap

- Introduction
- Stage 1: Instruction Fetch
- Stage 2: Instruction Decode
- Stage 3: Execution
- Stage 4-5: Data Memory, Write Back
- Evaluation
- Future Work/Conclusion

Future Work & Conclusion

- **Further improvements**
  - More testing
  - More optimizations (forwarding unit, caches etc)
- **Finalize the design**
  - Go through all the flow of generating the final file that is necessary for chip factories to build the chip (Clock tree, PlaceNRoute, etc.)

Future Work & Conclusion

With this work:

- We learned how to code in System Verilog and practically use it
- We learned how an implementation of the 32-bit RISC-V ISA can conduct all the needed information when executing an instruction
- We learned how to use Synopsys tools, in order to build our project (compiler, vcs simulator etc)

Questions?

Thank you!