MIPS 16 Bit Processor

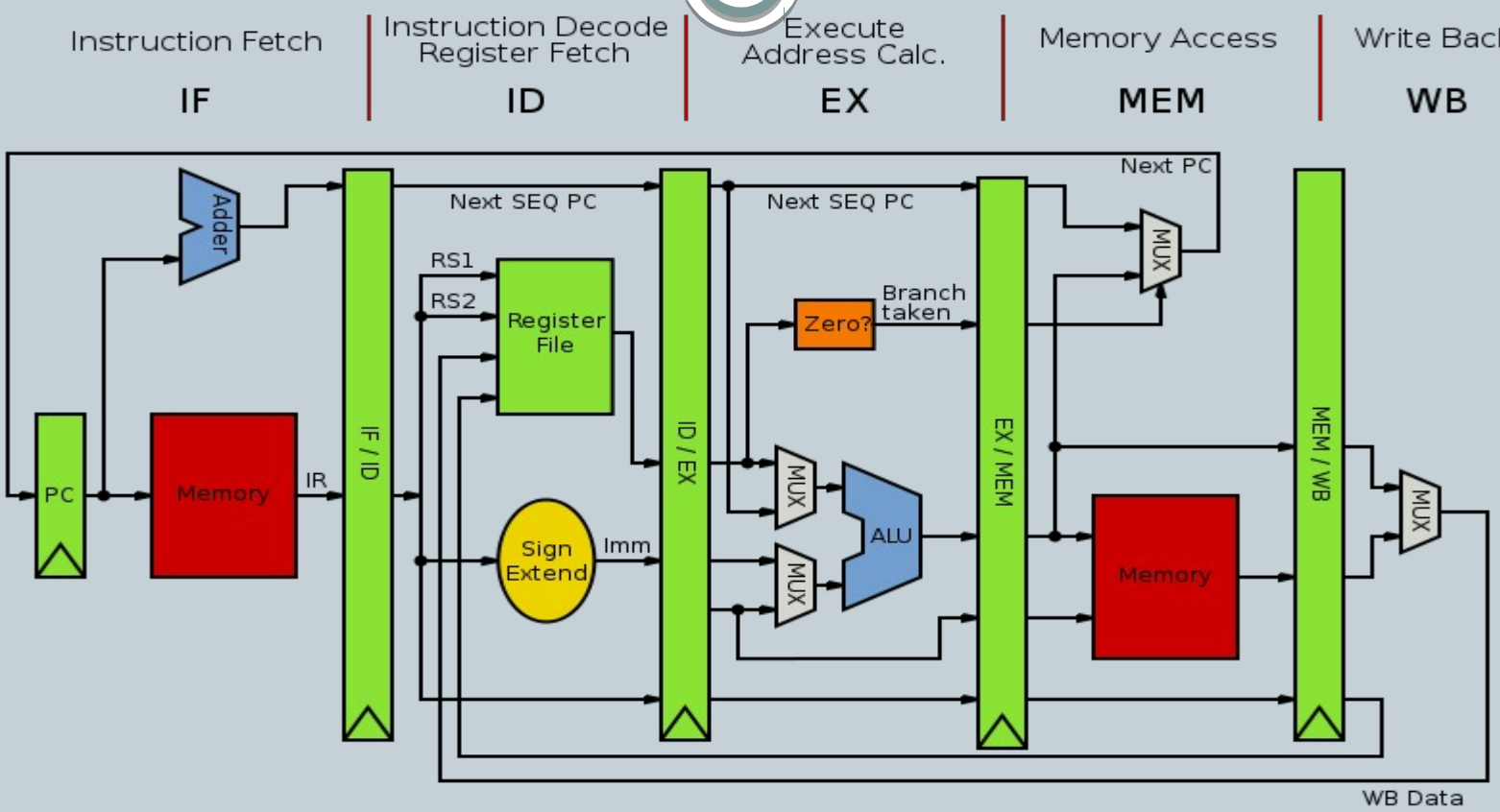
BY:

ROBERT MAXWELL
&
RYAN AGUDELO

MIPS Machine

- Microprocessor without Interlocked Pipeline Stages (MIPS)
- Reduced instruction set computer (RISC)
  - Simplified instructions can provide higher performance if the simplicity enables faster execution of each instruction
- Instruction set architecture (ISA)
  - Architecture related to programming
    - Native data types
    - Instructions
    - Registers
    - Addressing modes

MIPS Diagram

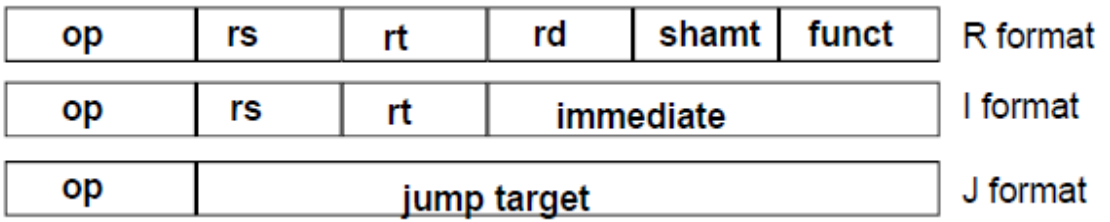


Pipelined MIPS, showing the five stages (instruction fetch, instruction decode, execute, memory access and write back)

Instruction Set

Assumptions:

- Subset of MIPS 1 ISA
- Byte/Word = 8/16 bits
- All instructions have
  R (register), I (immediate) or
  J (jump) format.
- No Floating Point Co-Processor
- Minimum instruction (256 bytes and data memory(512 bytes)



Arithmetic Operations			
ADD	Rd, Rs, Rt	Rd = Rs + Rt	(overflow trap)
ADDI	Rd, Rs, const16	Rd = Rs + const16$^{\pm}$	(overflow trap)
ADDIU	Rd, Rs, const16	Rd = Rs + const16$^{\pm}$	
ADDU	Rd, Rs, Rt	Rd = Rs + Rt	
LUI	Rd, const16	Rd = const16 << 16	
SUB	Rd, Rs, Rt	Rd = Rs – Rt	(overflow trap)
SUBU	Rd, Rs, Rt	Rd = Rs – Rt	

Logical and Bit-Field Operations		
AND	Rd, Rs, Rt	Rd = Rs & Rt
ANDI	Rd, Rs, const16	Rd = Rs & const16$^{\varnothing}$
NOP		No-op
OR	Rd, Rs, Rt	Rd = Rs | Rt
ORI	Rd, Rs, const16	Rd = Rs | const16$^{\varnothing}$

Condition Testing and Conditional Move Operations		
SLT	Rd, Rs, Rt	Rd = (Rs$^{\pm}$ < Rt$^{\pm}$) ? 1 : 0
SLTI	Rd, Rs, const16	Rd = (Rs$^{\pm}$ < const16$^{\pm}$) ? 1 : 0
SLTIU	Rd, Rs, const16	Rd = (Rs$^{\varnothing}$ < const16$^{\varnothing}$) ? 1 : 0
SLTU	Rd, Rs, Rt	Rd = (Rs$^{\varnothing}$ < Rt$^{\varnothing}$) ? 1 : 0

Shift and Rotate Operations		
SLL	Rd, Rs, shift5	Rd = Rs << shift5
SRL	Rd, Rs, shift5	Rd = Rs$^{\varnothing}$ >> shift5

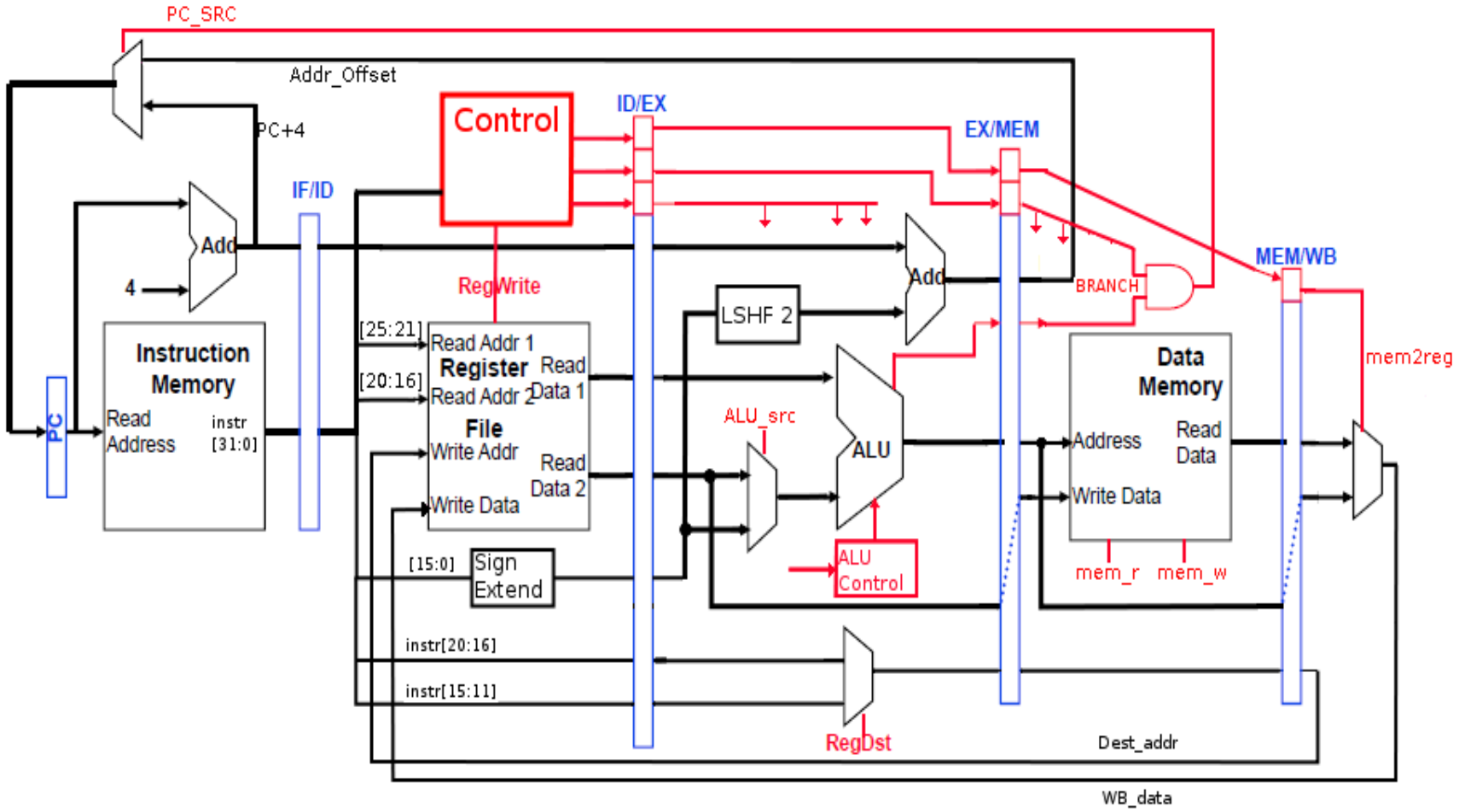
Load and Store Operations		
LW	Rd, off16(Rs)	Rd = mem32(Rs + off16$^{\pm}$)
SW	Rs, off16(Rt)	mem32(Rt + off16$^{\pm}$) = Rs

Jumps And Branches (Note: One Delay Slot)		
BEQ	Rs, Rt, off18	if Rs = Rt, PC += off18$^{\pm}$
BEQZ	Rs, off18	if Rs = 0, PC += off18$^{\pm}$
BGTZ	Rs, off18	if Rs > 0, PC += off18$^{\pm}$
BLTZ	Rs, off18	if Rs < 0, PC += off18$^{\pm}$
BNE	Rs, Rt, off18	if Rs ≠ Rt, PC += off18$^{\pm}$

Verilog-HDL Design Procedure

- Code the initial hardware and control signals
- Test each arithmetic state to verify appropriate functionality
- Test Branch functionality
- Alter design (if needed – then re-test)
- Test for correctness of data
- Alter design (if needed – then re-test)
- Test the circuit with a program that proves correct operation

Initial Design

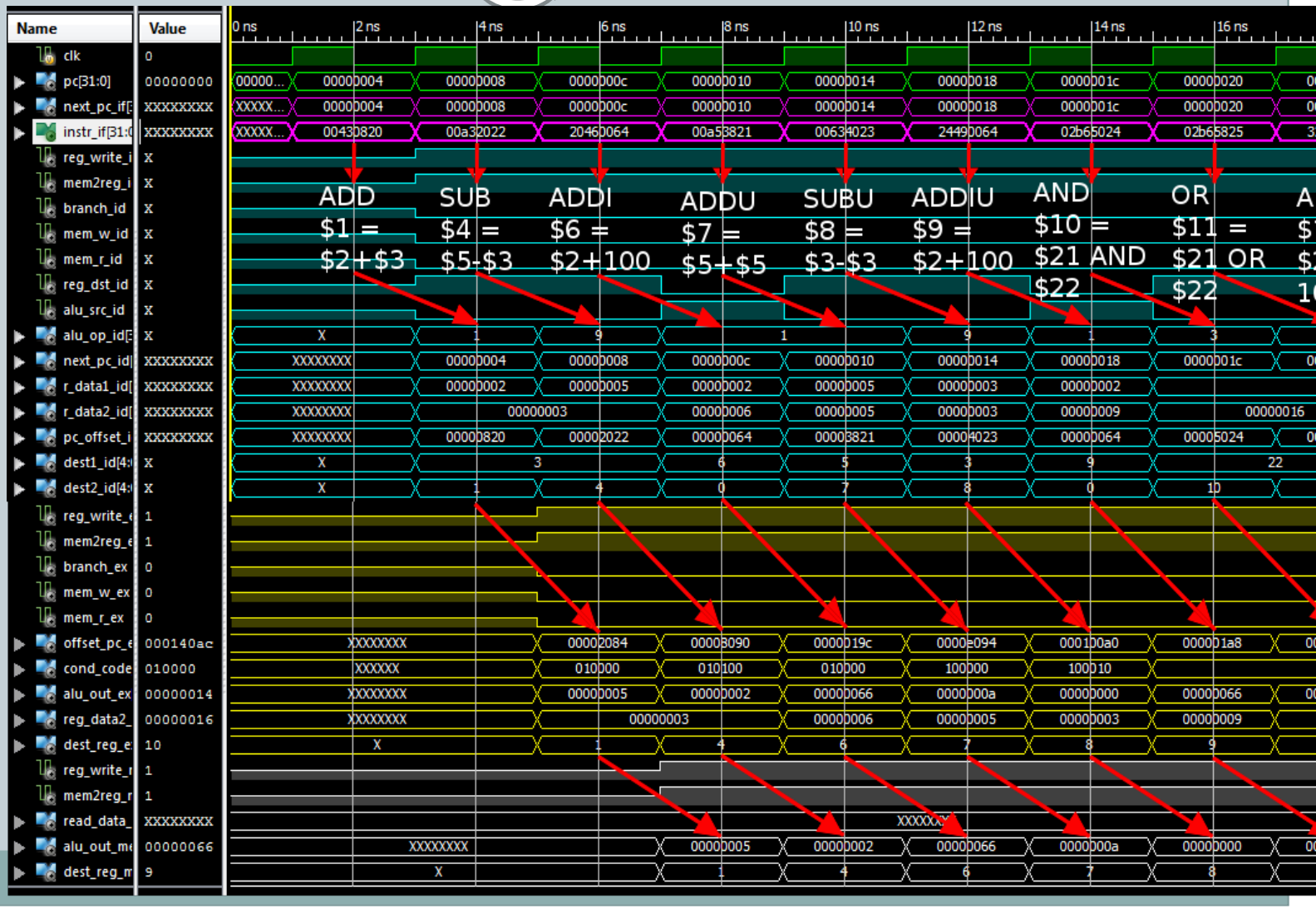


Courtesy of Dr. Byeong Lee : Microcomputer Architecture Lectures (UTSA)

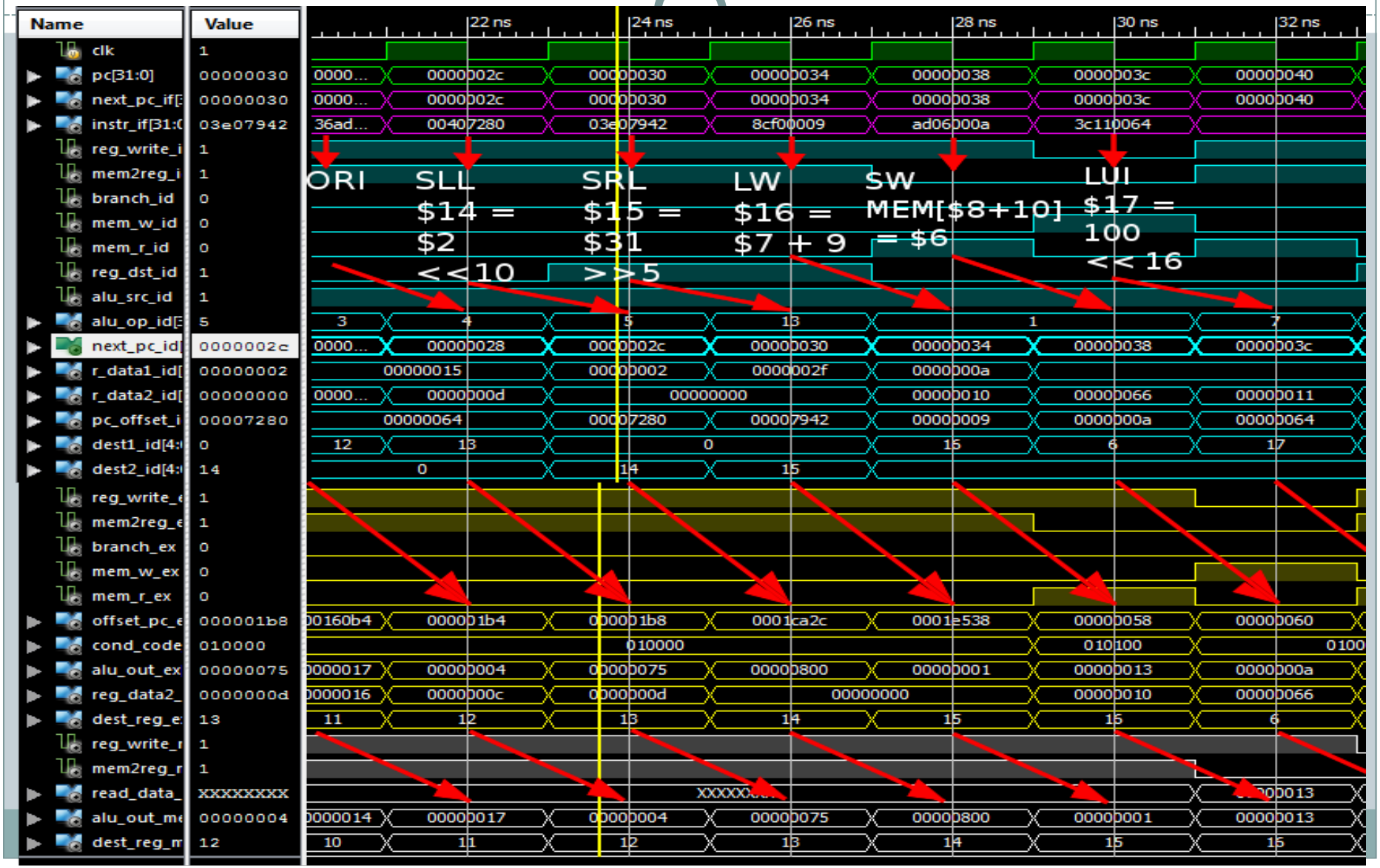
Test Code (Arithmetic)

Instruction	Instruction Code	Expected Result
ADD $1,$2,$3	0x00430820	R1 = 0x00000005
SUB $4,$5,$3	0x00A32022	R4 = 0x00000002
ADDI $6,$2,100	0x20460064	R6 = 0x00000066
ADDU $7,$5,$5	0x00A53821	R7 = 0x0000000A
SUBU $8,$3,$3	0x00634023	R8 = 0x00000000
ADDIU $9,$2,100	0x24490064	R9 = 0x00000066
AND $10,$21,$22	0x02B65024	R10=0x00000014
OR $11,$21,$22	0x02B65825	R11=0x00000017
ANDI $12,$21,100	0x32AC0064	R12=0x00000004
ORI $13,$21,100	0x36AD0064	R13=0x00000075
SLL $14,$2,10	0x00407280	R14=0x00000800
SRL $15,$31,5	0x03E07942	R15=0x00000001
LW $16,$7,9	0x8CF00009	R16=0x00000013
SW $6,$8,10	0xAD06000A	MEM[100] = 0x66
LUI $17,100	0x3C110064	R17=0x00640000

ADD to AND (Immed$_{16}$)



OR (Immed$_{16}$) to Load Upper (Immed$_{16}$)



Initial Results

Expected Result

R1 = 0x00000005

R4 = 0x00000002

R6 = 0x00000066

R7 = 0x0000000A

R8 = 0x00000000

R9 = 0x00000066

R10=0x00000014

R11=0x00000017

R12=0x00000004

R13=0x00000075

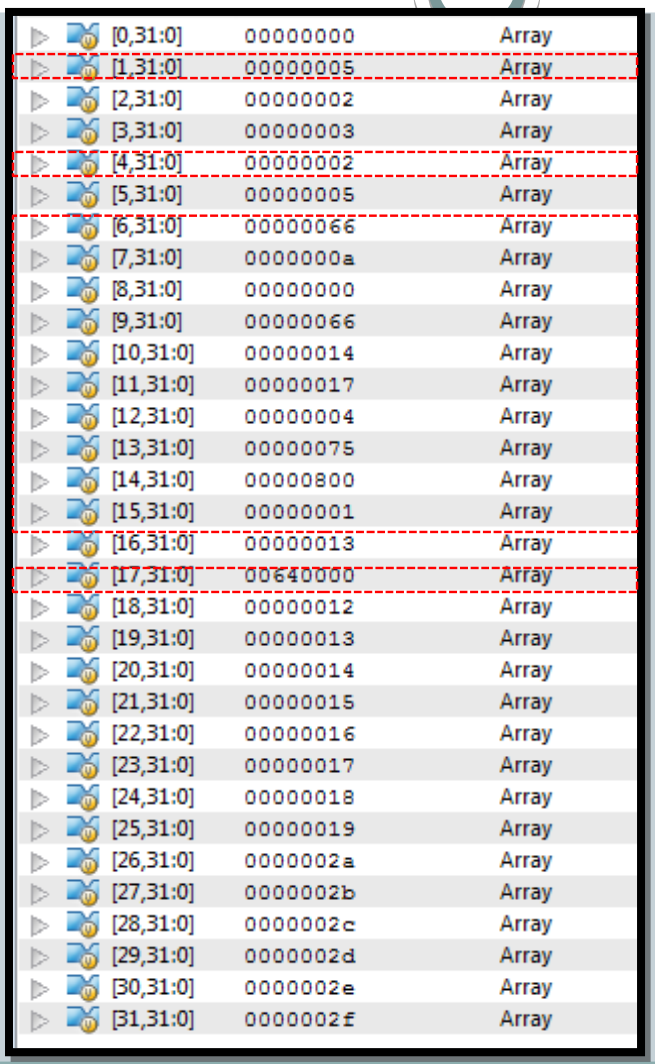
R14=0x00000800

R15=0x00000001

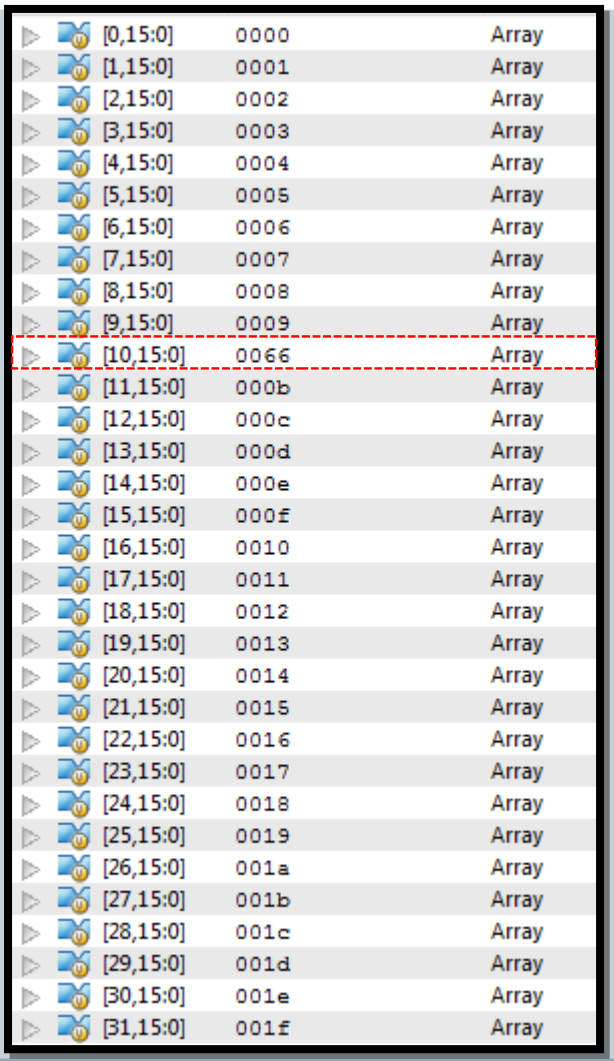
R16=0x00000013

MEM[100] = 0x66

R17=0x00640000



[0,31:0]	00000000	Array
[1,31:0]	00000005	Array
[2,31:0]	00000002	Array
[3,31:0]	00000003	Array
[4,31:0]	00000002	Array
[5,31:0]	00000005	Array
[6,31:0]	00000066	Array
[7,31:0]	0000000a	Array
[8,31:0]	00000000	Array
[9,31:0]	00000066	Array
[10,31:0]	00000014	Array
[11,31:0]	00000017	Array
[12,31:0]	00000004	Array
[13,31:0]	00000075	Array
[14,31:0]	00000800	Array
[15,31:0]	00000001	Array
[16,31:0]	00000013	Array
[17,31:0]	00640000	Array
[18,31:0]	00000012	Array
[19,31:0]	00000013	Array
[20,31:0]	00000014	Array
[21,31:0]	00000015	Array
[22,31:0]	00000016	Array
[23,31:0]	00000017	Array
[24,31:0]	00000018	Array
[25,31:0]	00000019	Array
[26,31:0]	0000002a	Array
[27,31:0]	0000002b	Array
[28,31:0]	0000002c	Array
[29,31:0]	0000002d	Array
[30,31:0]	0000002e	Array
[31,31:0]	0000002f	Array

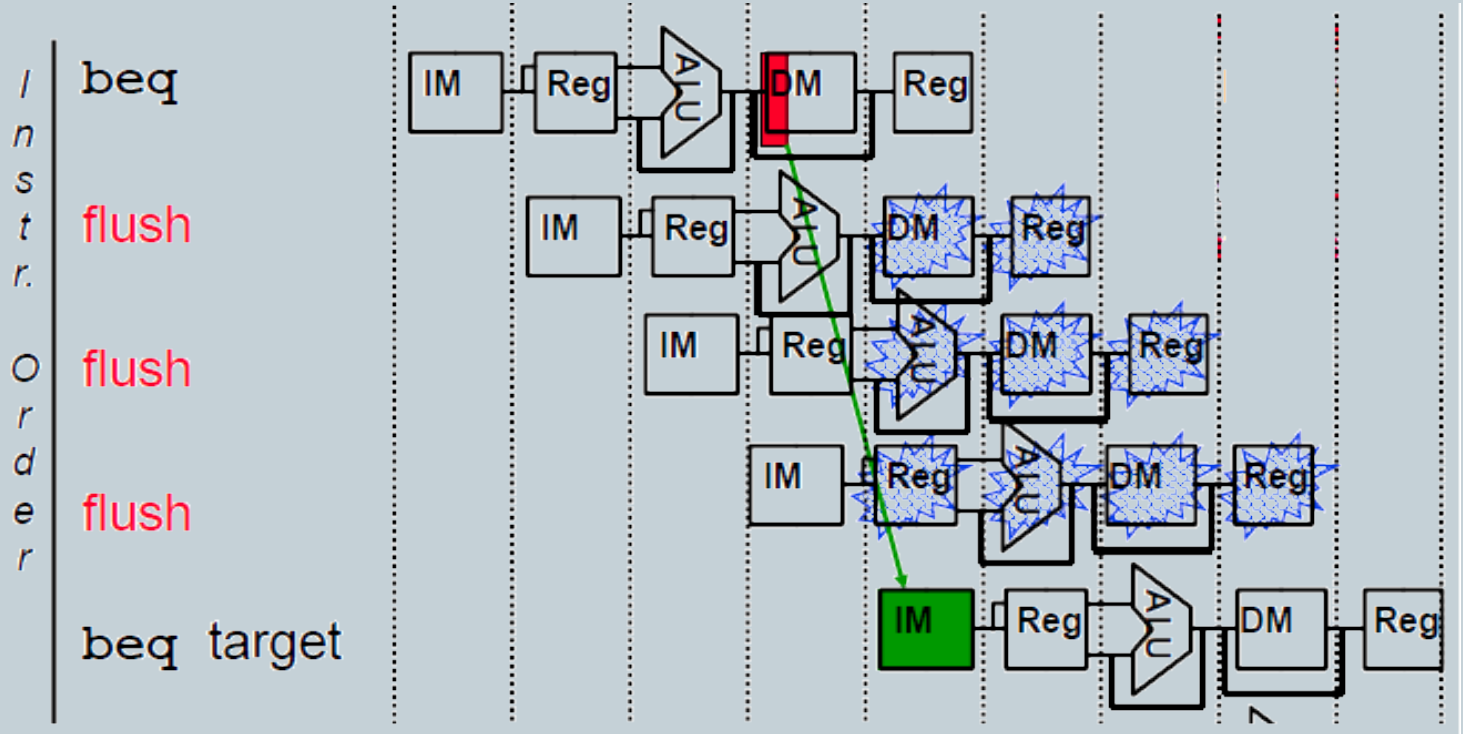


[0,15:0]	0000	Array
[1,15:0]	0001	Array
[2,15:0]	0002	Array
[3,15:0]	0003	Array
[4,15:0]	0004	Array
[5,15:0]	0005	Array
[6,15:0]	0006	Array
[7,15:0]	0007	Array
[8,15:0]	0008	Array
[9,15:0]	0009	Array
[10,15:0]	0066	Array
[11,15:0]	000b	Array
[12,15:0]	000c	Array
[13,15:0]	000d	Array
[14,15:0]	000e	Array
[15,15:0]	000f	Array
[16,15:0]	0010	Array
[17,15:0]	0011	Array
[18,15:0]	0012	Array
[19,15:0]	0013	Array
[20,15:0]	0014	Array
[21,15:0]	0015	Array
[22,15:0]	0016	Array
[23,15:0]	0017	Array
[24,15:0]	0018	Array
[25,15:0]	0019	Array
[26,15:0]	001a	Array
[27,15:0]	001b	Array
[28,15:0]	001c	Array
[29,15:0]	001d	Array
[30,15:0]	001e	Array
[31,15:0]	001f	Array

- Register File
- Memory

Branch Hazards

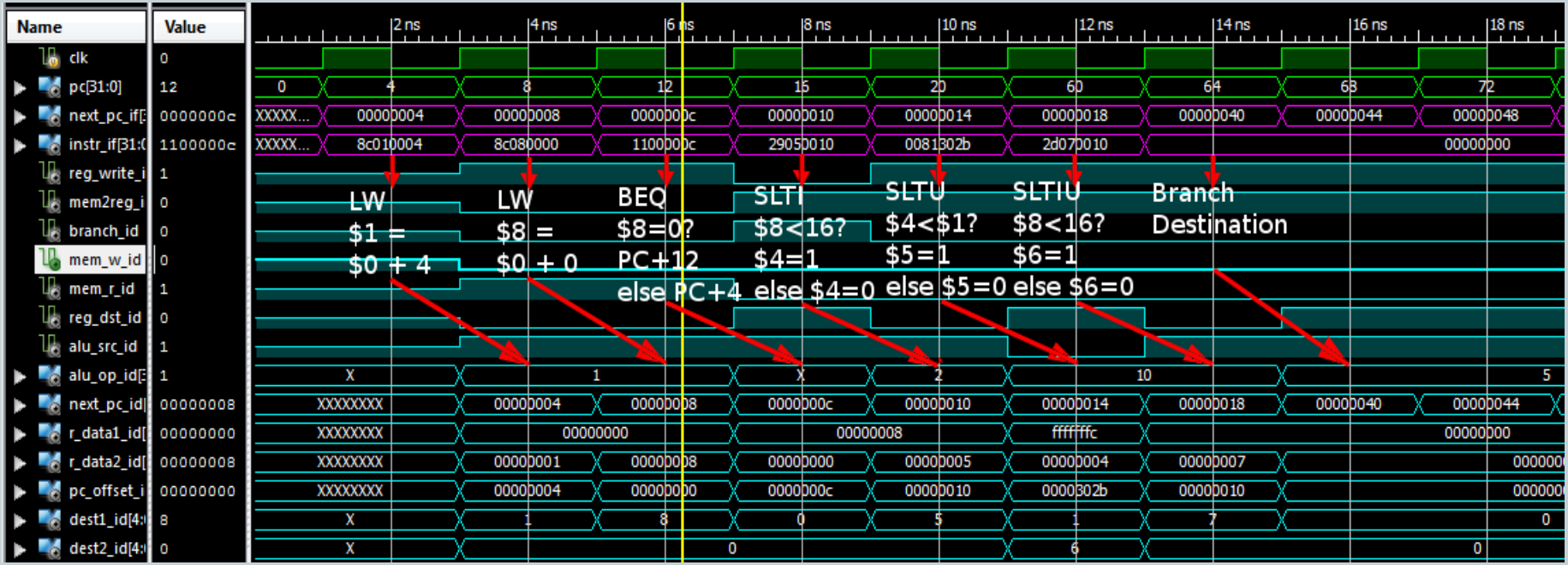
- If the Branch is taken, next 3 instructions must be burned and flushed from the Pipeline



Courtesy of Dr. Byeong Kil Lee : Microcomputer Architecture Lectures (UTSA)

Branch Hazard Example

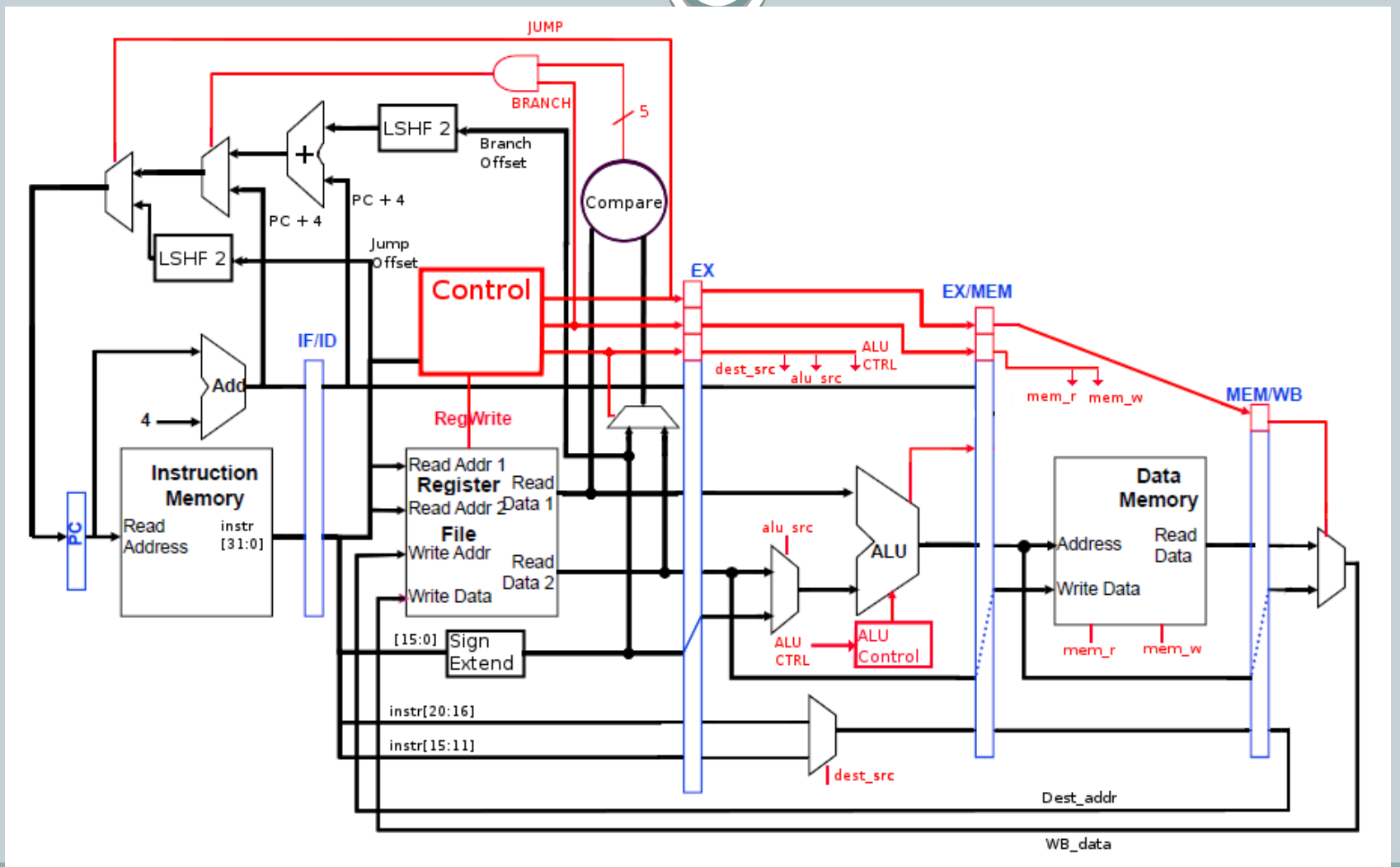
- Branch destination gets loaded 4 cycles after decision



Solutions

- Branch Prediction – Many options possible (predict taken, not taken, 2-bit predictor, other algorithms[2])
  - For this design, we predict not taken
- Move branch decision hardware to the ID stage so as to only incur a 1 cycle penalty

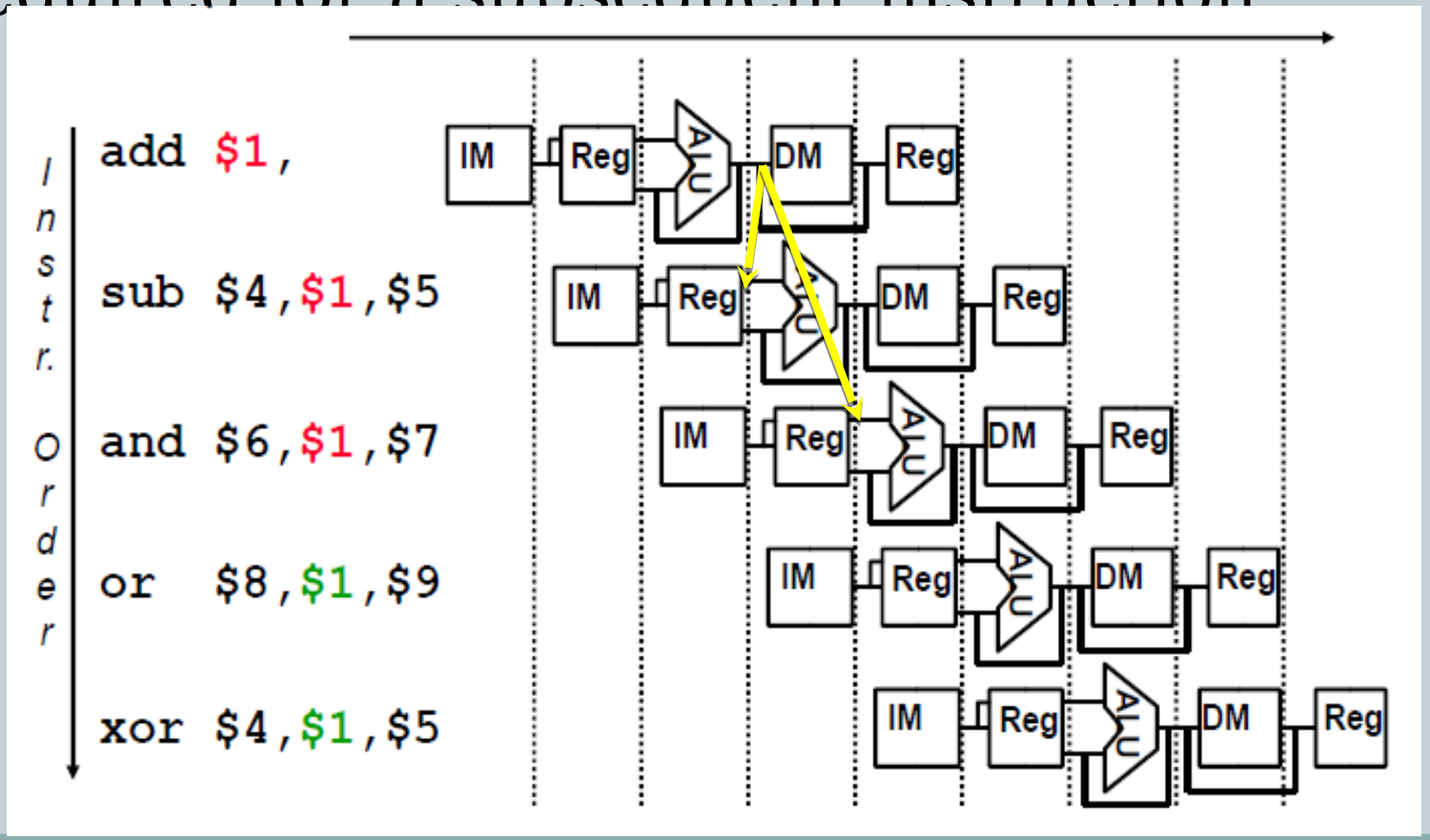
Branch Decision to ID Stage



Adapted from Dr. Byeong Kil Lee : Microcomputer Architecture Lectures (UTSA)

Data Hazards

- Occurs when the result of an instruction is required for a subsequent instruction



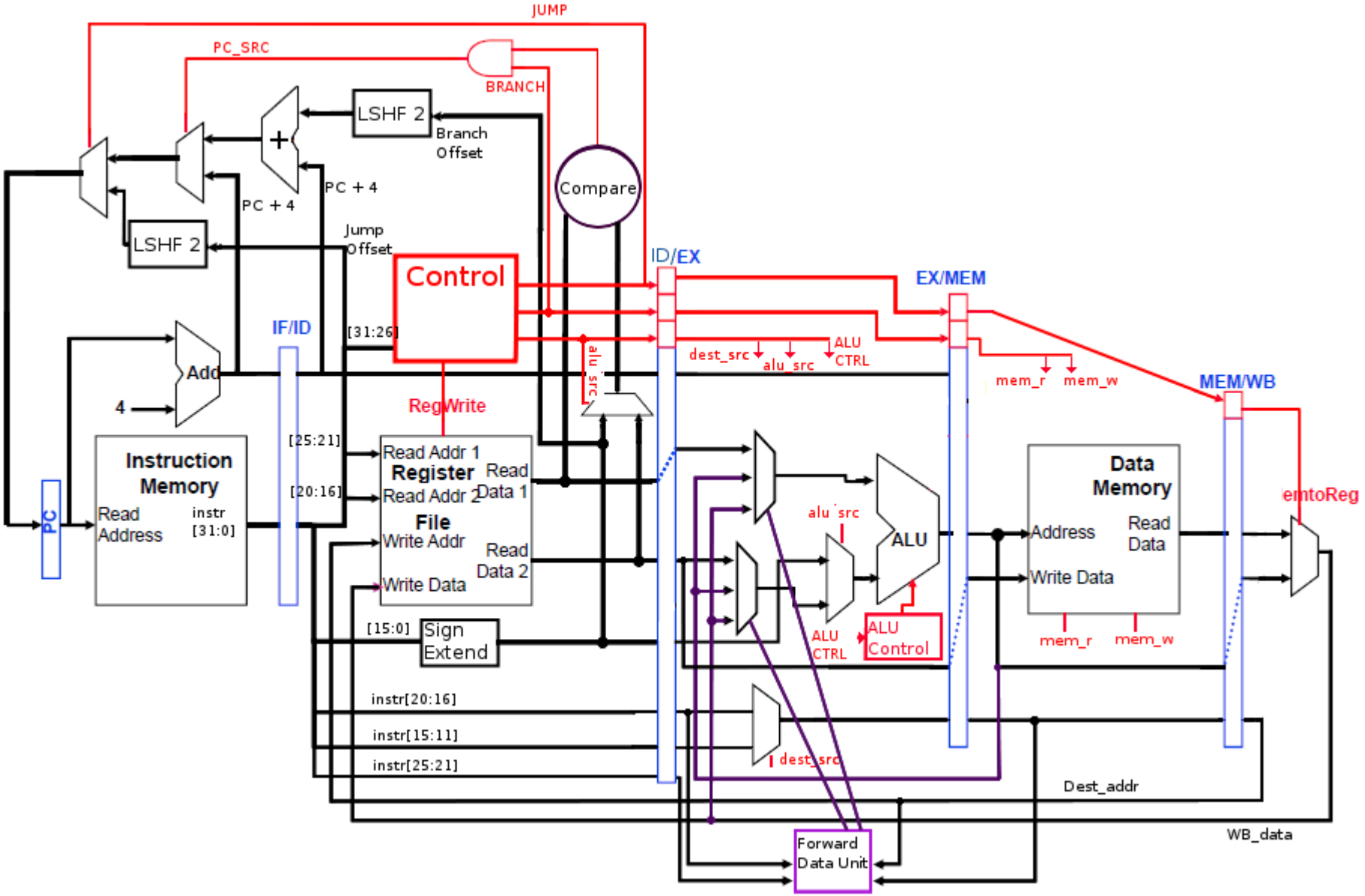
Courtesy of Dr. Byeong Kil Lee : Microcomputer Architecture Lectures (UTSA)

Solution

- Add hardware to the EX stage that tests the destination and source register addresses to determine if the result of an operation is needed by the next few instructions.

- Here we compare source1 (EX stage) with source2 (EX stage) with reg_dest (MEM stage) and reg_dest (WB stage)

Completed Design



Final Design Test Code

- Takes sum first ‘n’ Fibonacci numbers and saves to memory[3]

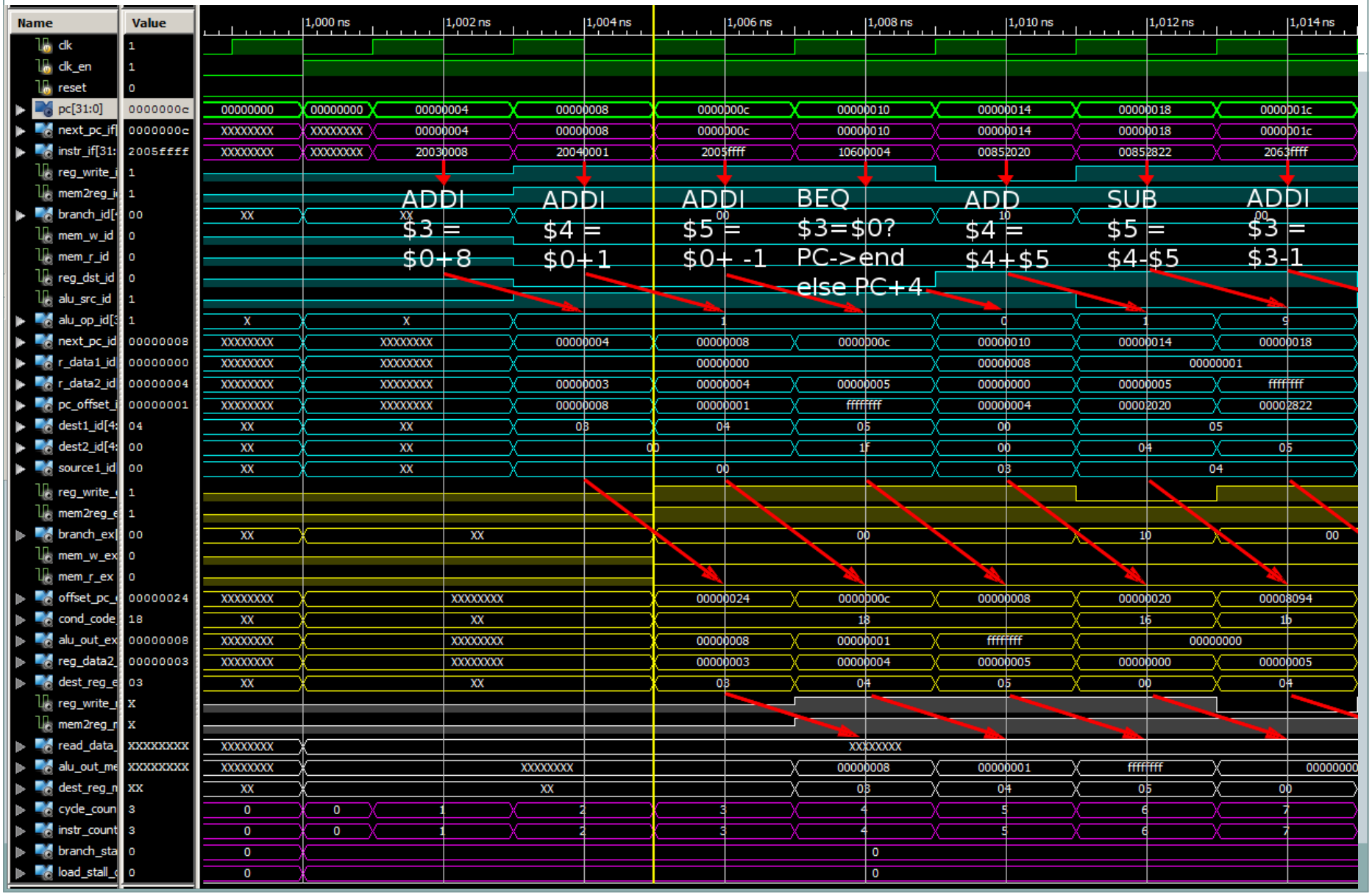
HEX Code

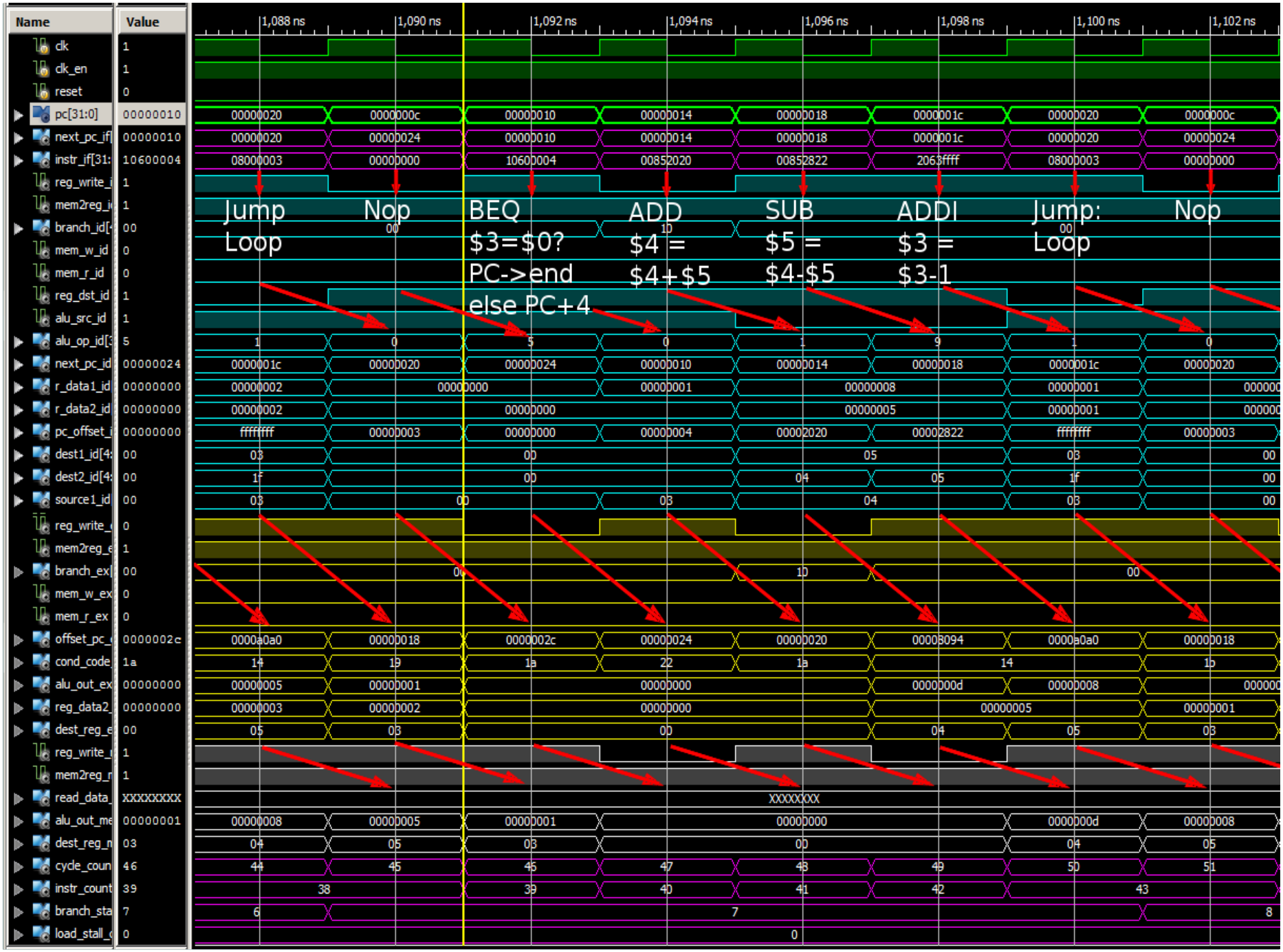
```
Fib:  addi  $3,$0,8 // init $3 = n 0x20030008
    addi  $4,$0,1 //$4 = 1        0x20040001
    addi  $5,$0,-1    //$5 = -1   0x2005FFFF

Loop:    beq   $3,$0,end //if $3==0, branch    0x10600008
     add  $4,$4,$5   //$4 = $4 + $5  0x00852020
     sub  $5,$4,$5   //$5 = $4 - $5   0x00852822
     addi $3,$3,-1                  //$3 = $3 – 1  0x2063FFFF
     j  Loop // jump  0x0800000C
    nop   0x00000000

End:     stw   $4,32($0) //store final value  0xAC040020
```

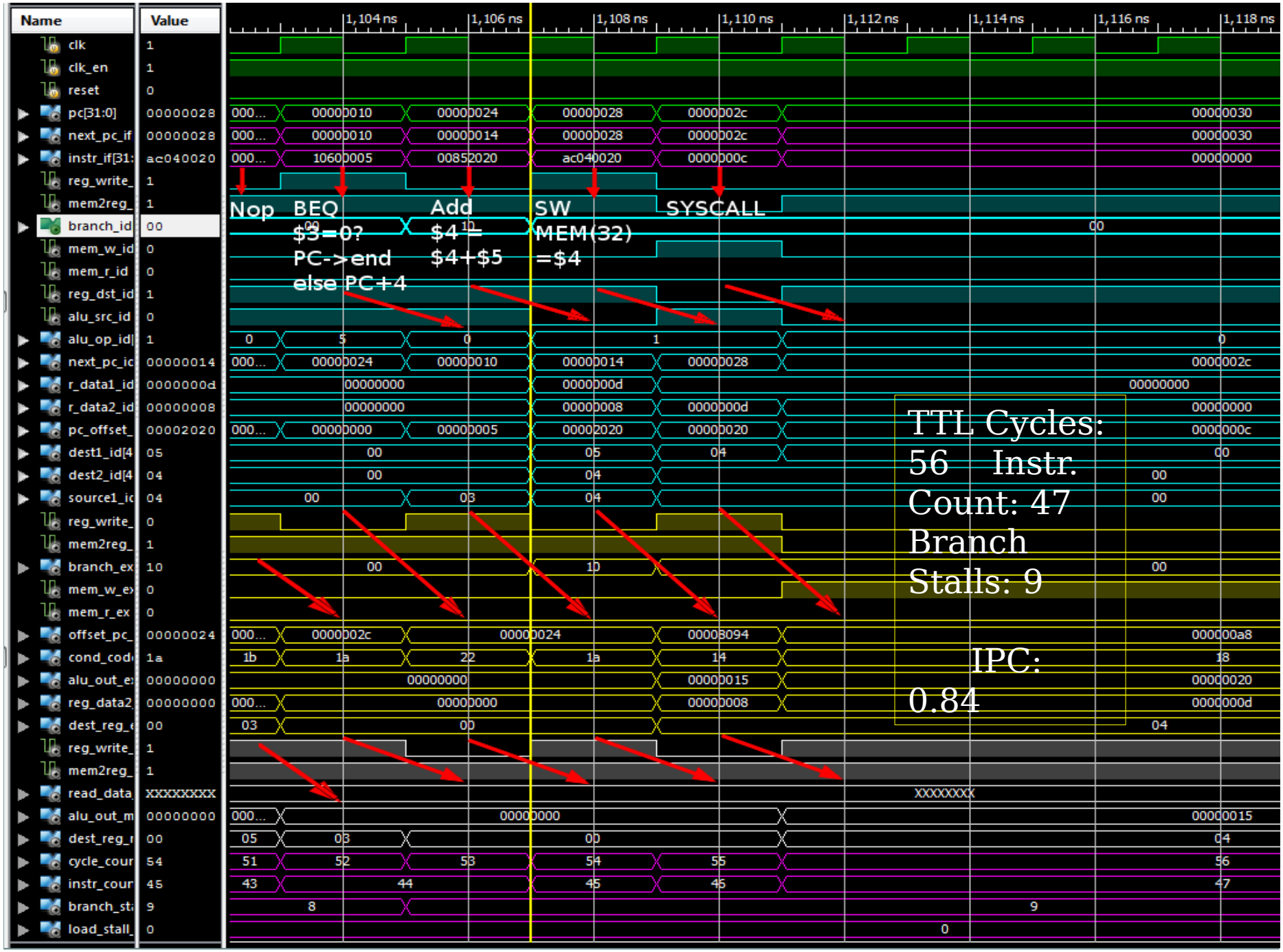
Expected Values

n	8	7	6	5	4	3	2	1	0
$3	8	7	6	5	4	3	2	1	0
$4	0	1	1	2	3	5	8	13	Break
$5	-1	1	0	1	1	2	3	5	8
MEM	Prev.	Prev.	Prev.	Prev.	Prev.	Prev.	Prev.	Prev.	13

Simulation Results

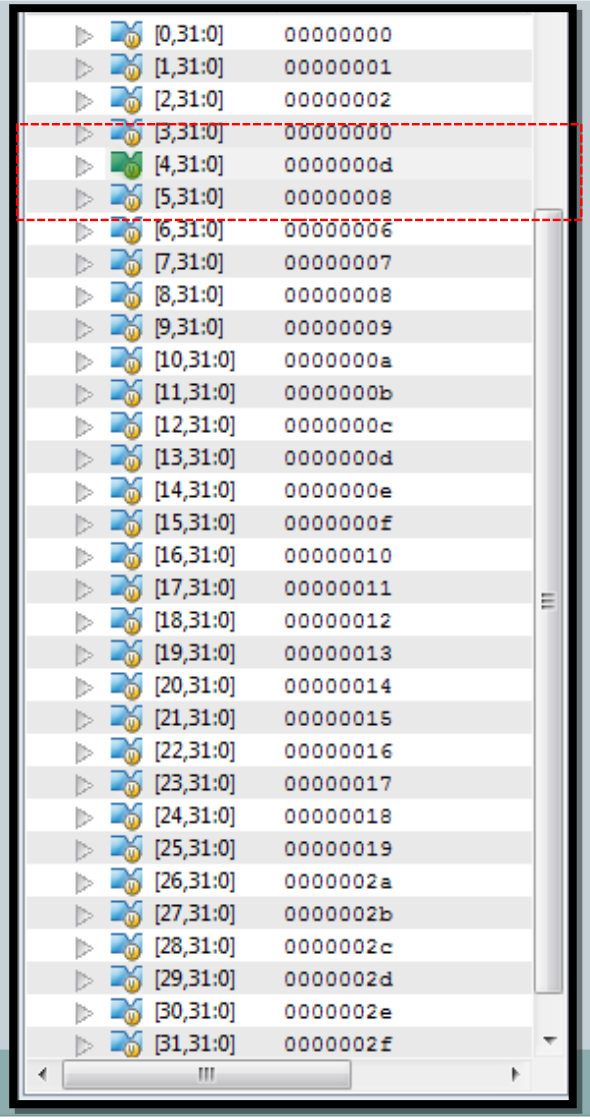



Name
Value
1,088 ns
1,090 ns
1,092 ns
1,094 ns
1,096 ns
1,098 ns
1,100 ns
1,102 ns
Jump Loop
Nop
BEQ $3=$0? PC->end else PC+4
ADD $4 = $4+$5
SUB $5 = $4-$5
ADDI $3 = $3-1
Jump: Loop
Nop

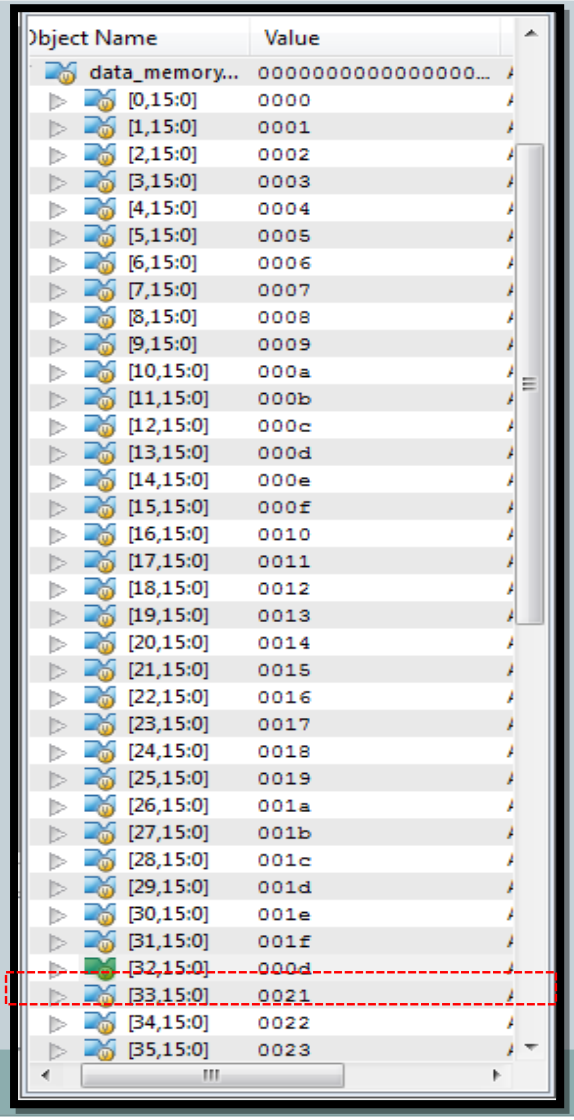

Name
Value
clk
clk_en
reset
pc[31:0]
next_pc_if
instr_if[31:
reg_write_
mem2reg_
branch_id
mem_w_id
mem_r_id
reg_dst_id
alu_src_id
alu_op_id
next_pc_id
r_data1_id
r_data2_id
pc_offset_
dest1_id[4
dest2_id[4
source1_id
reg_write_
mem2reg_
branch_ex
mem_w_ex
mem_r_ex
offset_pc_
cond_cod
alu_out_e
reg_data2
dest_reg_
reg_write_
mem2reg_
read_data
alu_out_m
dest_reg_
cycle_cour
instr_cour
branch_st
load_stall_
1,104 ns
1,106 ns
1,108 ns
1,110 ns
1,112 ns
1,114 ns
1,116 ns
1,118 ns
Nop
BEQ
$3=0?
PC->end
else PC+4
Add
$4=
$4+$5
SW
MEM(32)
=$4
SYSCALL
TTL Cycles:
56 Instr.
Count: 47
Branch
Stalls: 9
IPC:
0.84

Memory Results

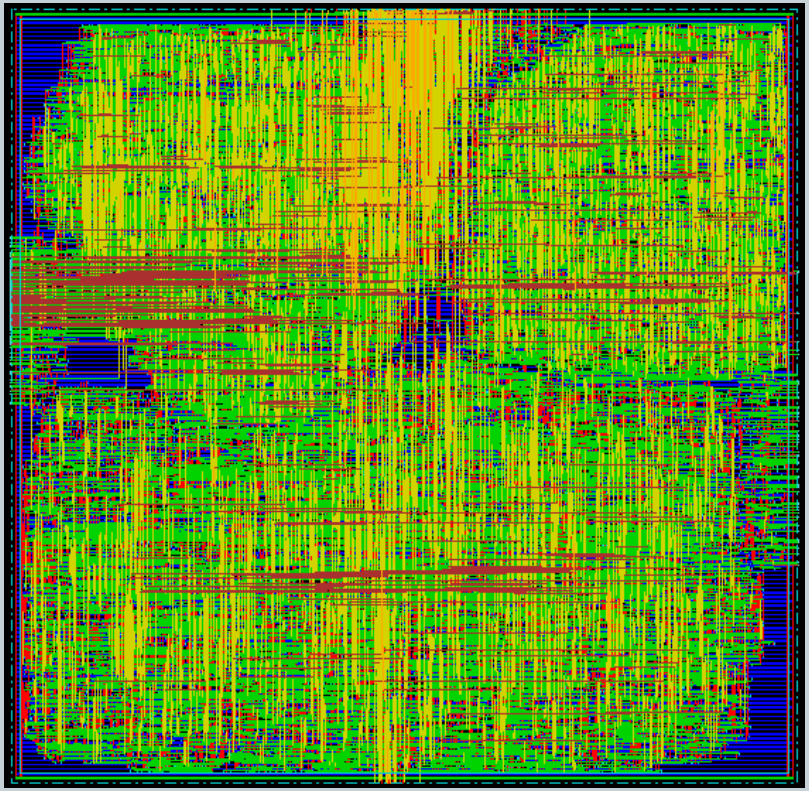
Register File

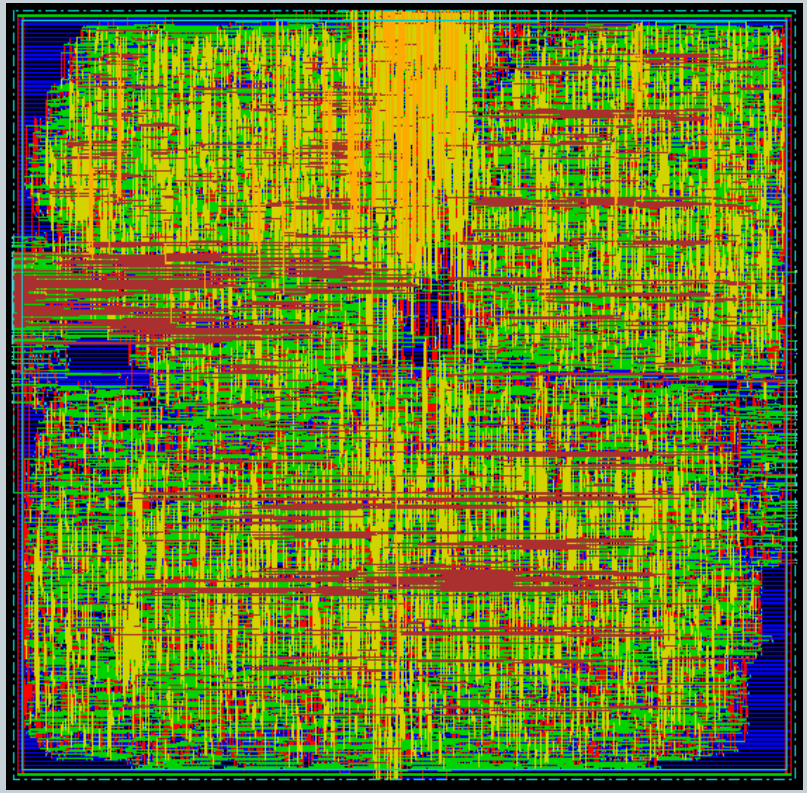


Data Memory



Encounter Layout

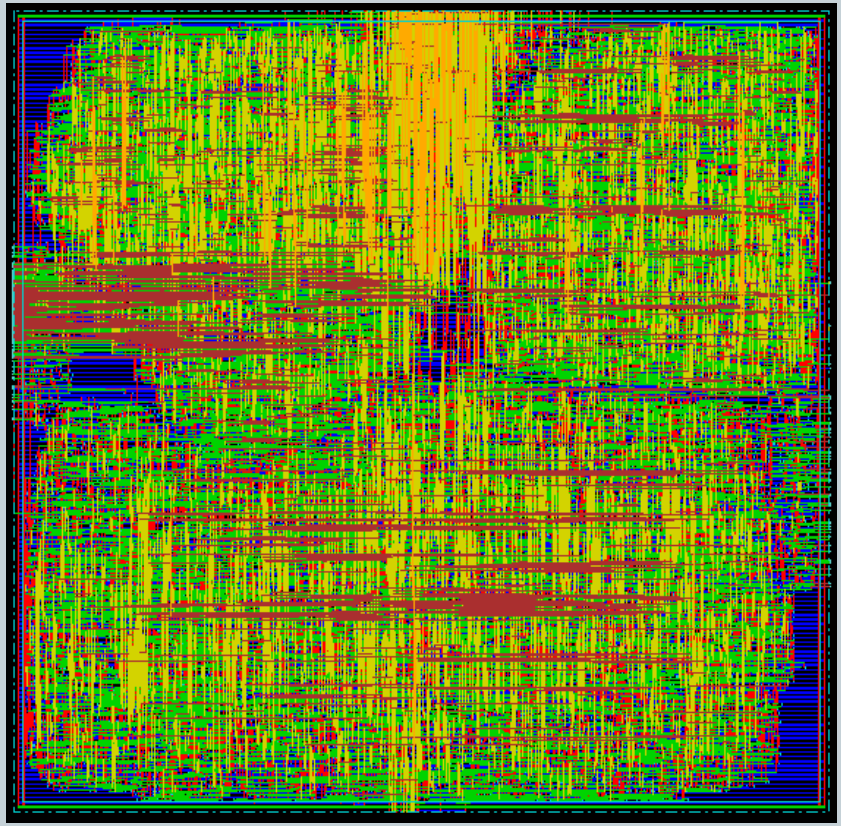




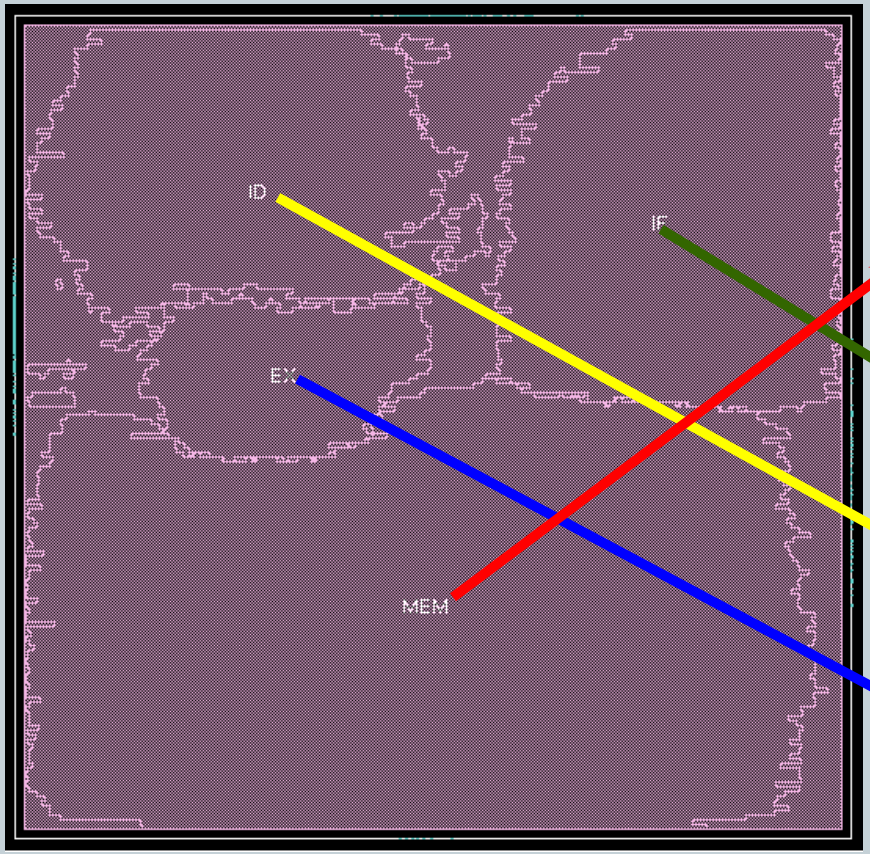
- Pre Nanoroute
- Post Nanoroute

Cadence Encounter

- Area
  - Chip Area: 2.82 x 2.82cm
  - Core Area: 2.75 x 2.75cm
  - Standard Cell Area: 1.94 x 1.94cm
- Power
  - Internal: 26.26mW
  - Leakage: 8.604mW
  - Total: 34.88mW
  - Switching: 0.01144mW
- Timing
  - WNS : -26.113ns
  - TNS: -1708.0ns



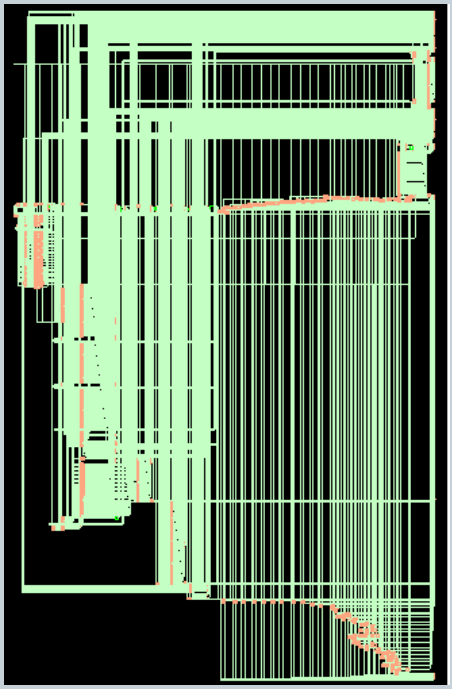
Cell Layout



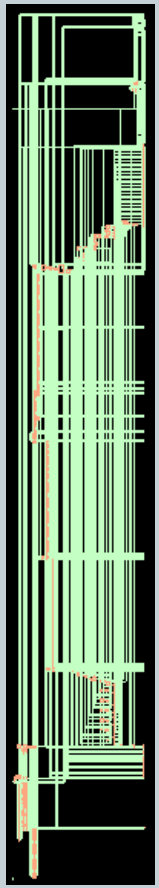
Instance	Cells	Cell Area	Net Area	Wireload
mips_16b	15551	375753	0	<none> (D)
MEM	7618	203965	0	<none> (D)
IF	3054	50432	0	<none> (D)
		34%		
		13%		
ID	2951	67815	0	<none> (D)
		18%		
EX	1103	19064	0	<none> (D)
		5%		

- Amoeba Space Allocation Layout

Schematics



- RTL Generated Schematic

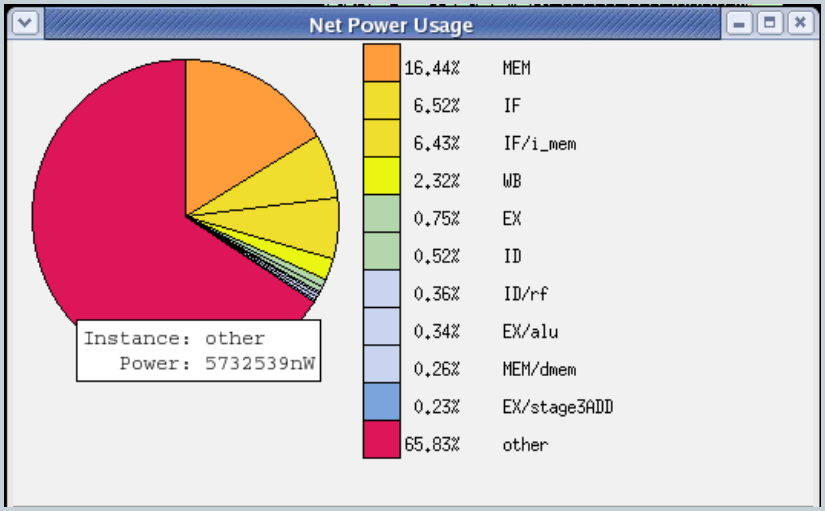


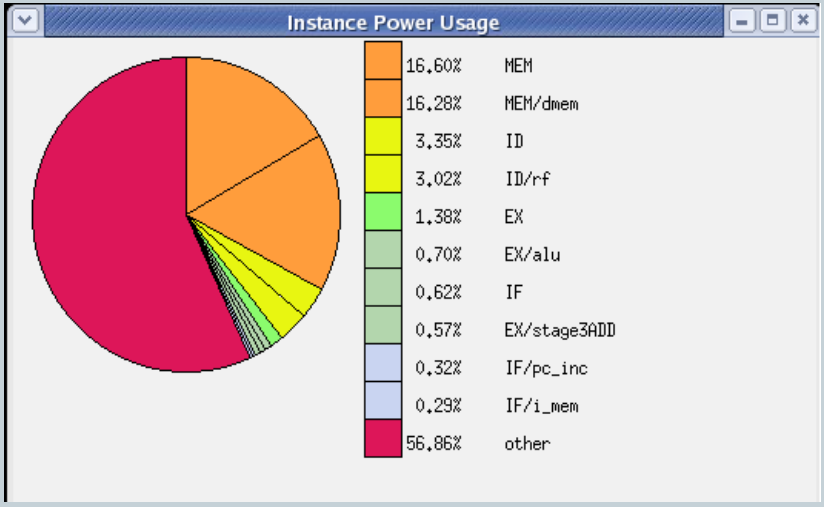
- Complete Encounter Generated Schematic

Power

Area of Power Net Distribution

Layer Name	Area of Power Net	Routable Area	Percentage
Metal1	60802.0000	756898.0524	8.0331%
Metal2	1536.8672	756898.0524	0.2030%
Metal3	782.8832	756898.0524	0.1034%
Metal4	0.0000	756898.0524	0.0000%
Metal5	0.0000	756898.0524	0.0000%
Metal6	0.0000	756898.0524	0.0000%





Compiler GUI Yielded some interesting details

bash% rc –f <gfile>.g -gui

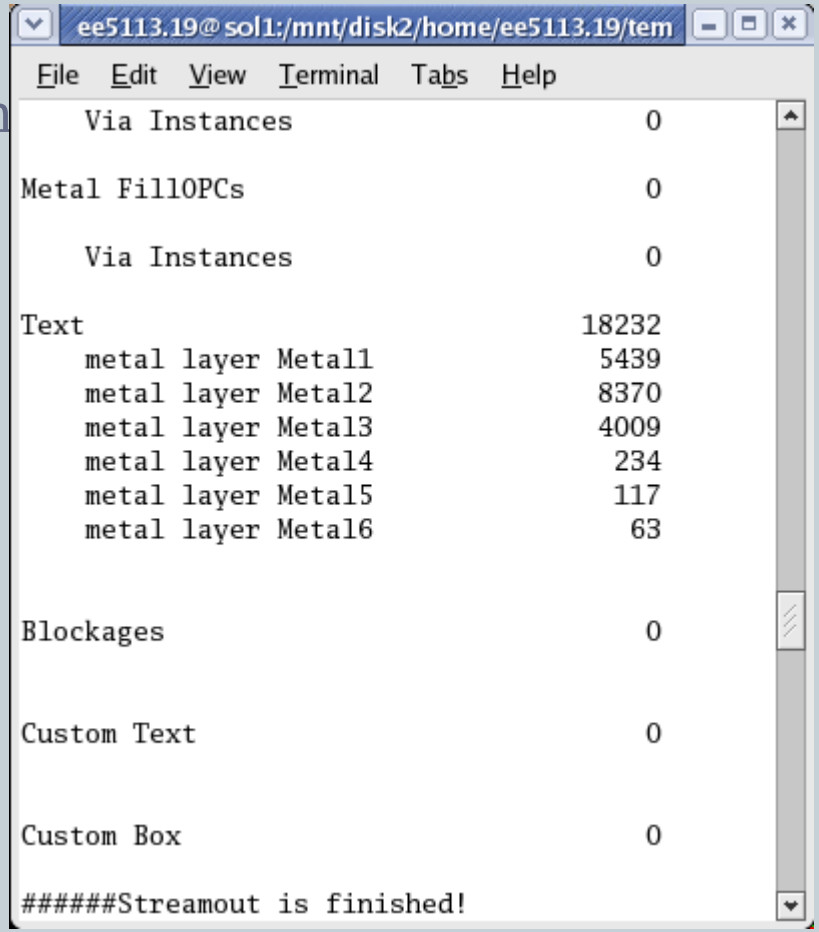
Physical Chip Characteristics

Wire Length Distribution

Total Metal1 wire length	46548.3450 um
Total Metal2 wire length	311237.6350 um
Total Metal3 wire length	372118.1150 um
Total Metal4 wire length	194577.3200 um
Total Metal5 wire length	71558.7500 um
Total Metal6 wire length	15791.4500 um
Total wire length	**1011831.6150 um ≈ 40 inches**
Average wire length/net	56.1629 um

Summary

- Prototype MIPS_16B:
  - Library: tsmc18 – 0.18um tech
  - Vdd: 1.8Vdc
  - Total Area: 2.82 x 2.82cm
  - Total Power: 34.88mW
  - Pin Count: 540 ext. pins



Questions?

Bibliography

[1] Dr. Byeong Lee, Computer Architecture Lectures, Lecture 3 – Pre_0,1,2,3 MIPS, University of Texas at San Antonio, Feb. 09, 2011.

[2] Hennessy, John and Patterson, David. Computer Architecture: A Quantitative Approach, 4th Edition. San Francisco: Elsevier,2007

[3] David Harris, Introduction to VLSI, Lecture 2: MIPS processor example, Harvey Mudd College, http://www.cmosvlsi.com/lect2.pdf, Spring 2004