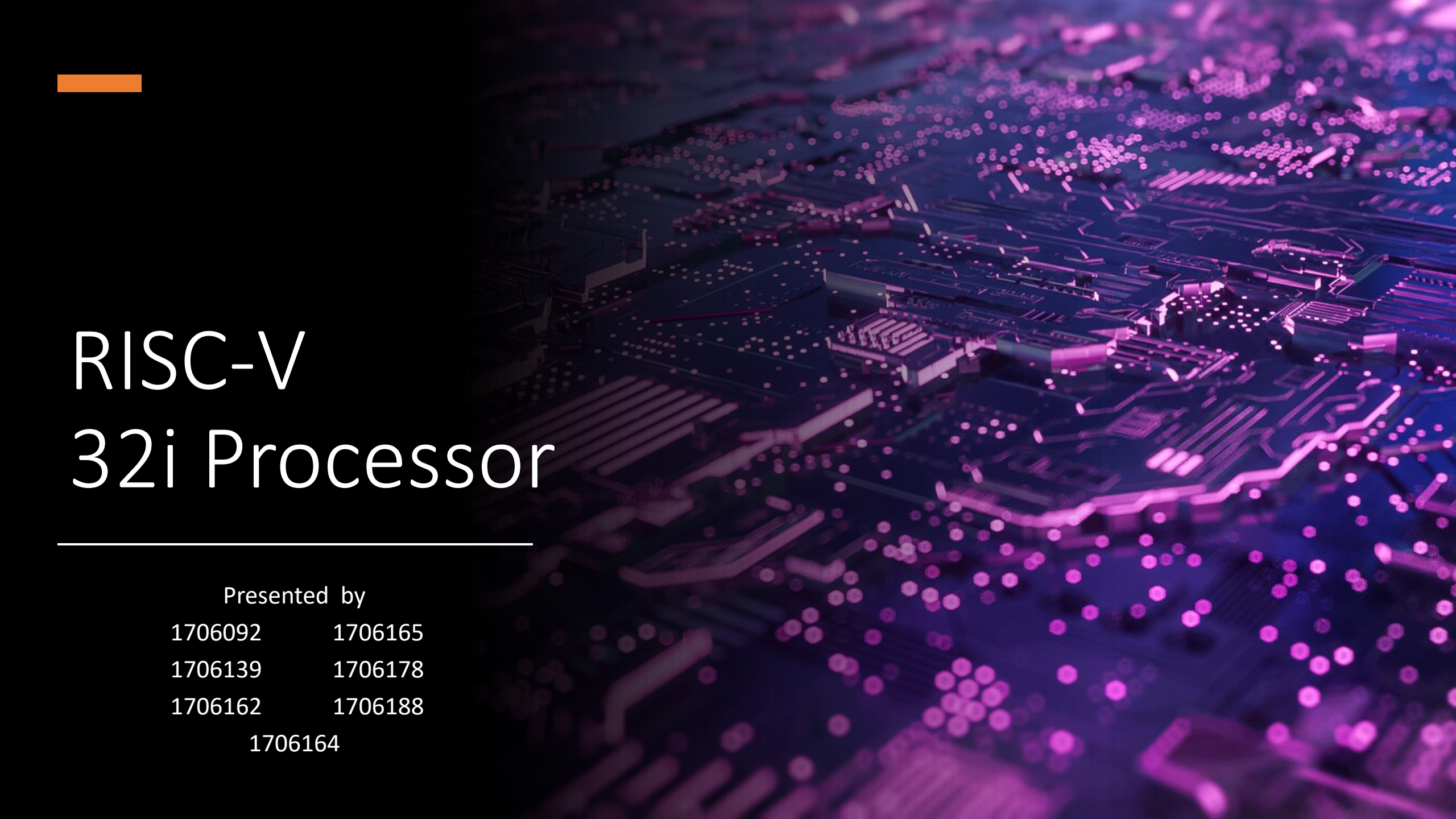


RISC-V 32i Processor

Presented by

1706092 1706165

1706139 1706178

1706162 1706188

1706164

INTRODUCTION

- RISC-V is an open-source instruction set architecture (ISA)
- It is a type of Reduced Instruction Set Computing (RISC) architecture
- It is designed to be simple, modular, and extensible
- RISC-V is highly scalable and customizable, making it suitable for a wide range of applications

SPECIFICATIONS

1.Core must be compatible with rv32i instruction set.
2. Implement base ISA spec. Reference: RISC-V ISA.
3. Use input global clock (positive edge triggered: clk), global reset (negative edge triggered: rst_n) and core_select signal
4. Functionality of core_select

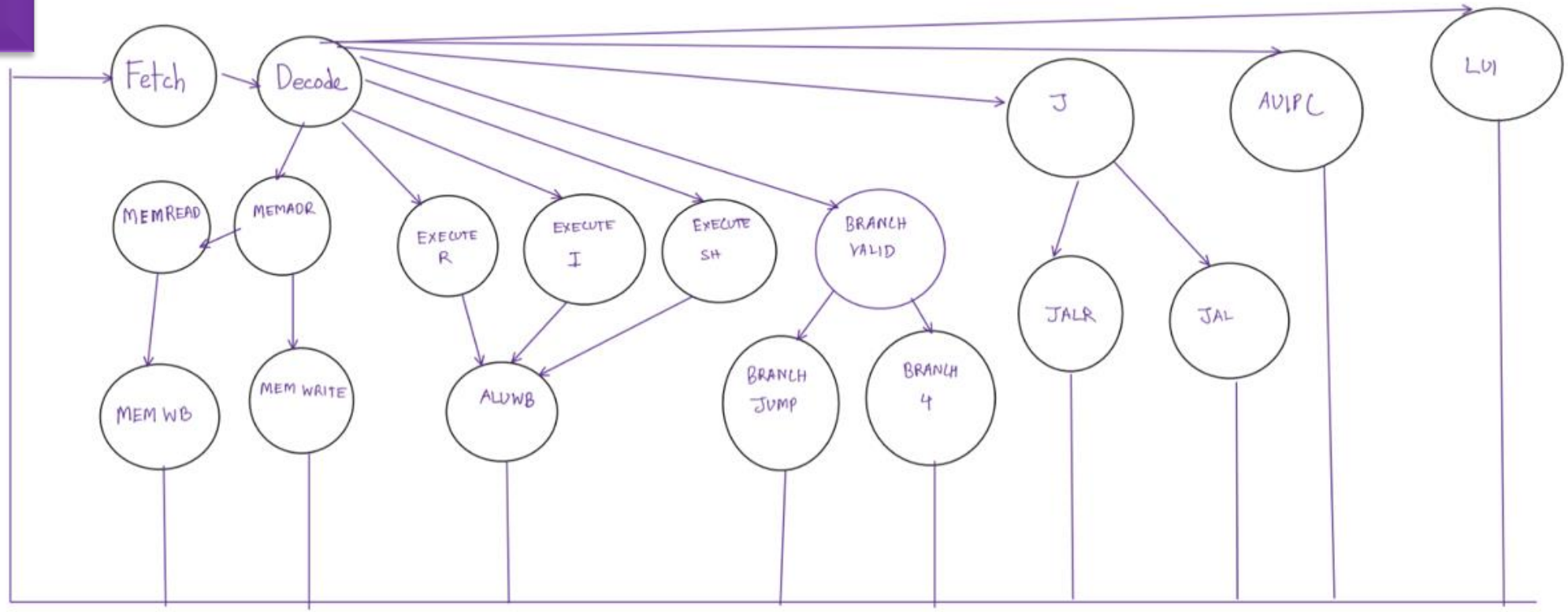
core_select	Functionality
0	1. Memory controller interface will be active to write data into the memory.
1	1. Core will start to work 2. Core side interface of the memory will be active to read and write data into the memory.

5. Core HDL descriptions need to be Synthesizable and optimized.
6. Base Clock Frequency 100 MHz

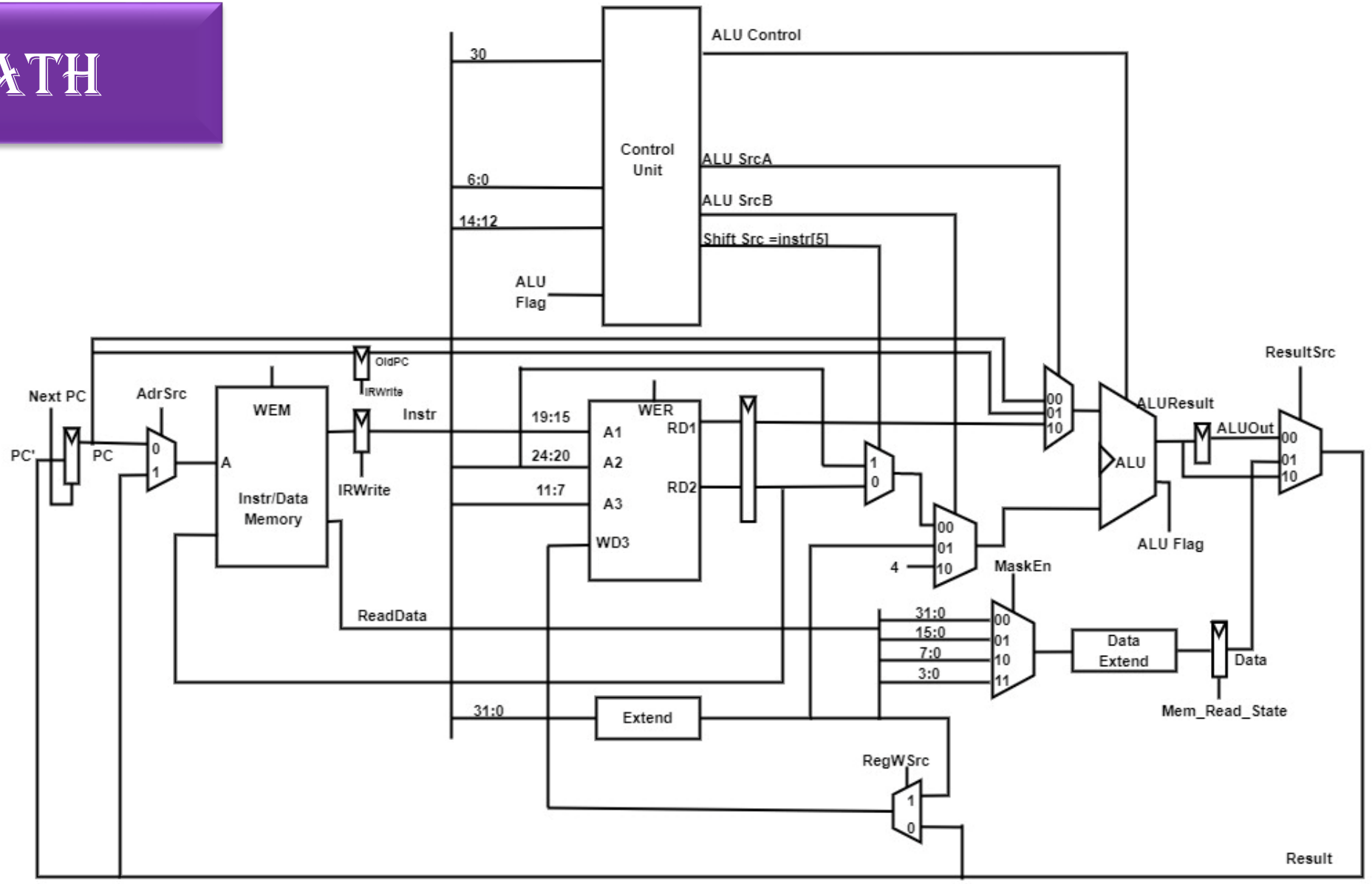
WHY MULTICYCLE?

- Faster Clock Speed Greater Flexibility
- Lower Hardware Costs.
- Improved Performance
- Reduced Power Consumption processor.

FSM


Fetch
Decode
MEMREAD
MEMADR
EXECUTE R
EXECUTE I
EXECUTE SH
BRANCH VALID
J
AUIPC
LUI
JALR
JAL
MEM WB
MEM WRITE
ALUWB
BRANCH JUMP
BRANCH 4

DATA PATH


ALU Control
30
Control Unit
6:0
14:12
ALU Flag
ALU SrcA
ALU SrcB
Shift Src =instr[5]
ResultSrc
OldPC
IRWrite
Next PC
AdrSrc
WEM
Instr
19:15
WER
ALUResult
ALUOut
PC'
PC
A
Instr/Data Memory
IRWrite
24:20
11:7
A1
A2
A3
WD3
RD1
RD2
ALU
ALU Flag
4
MaskEn
ReadData
31:0
15:0
7:0
3:0
Data Extend
Data
Mem_Read_State
31:0
Extend
RegWSrc
Result

CPI

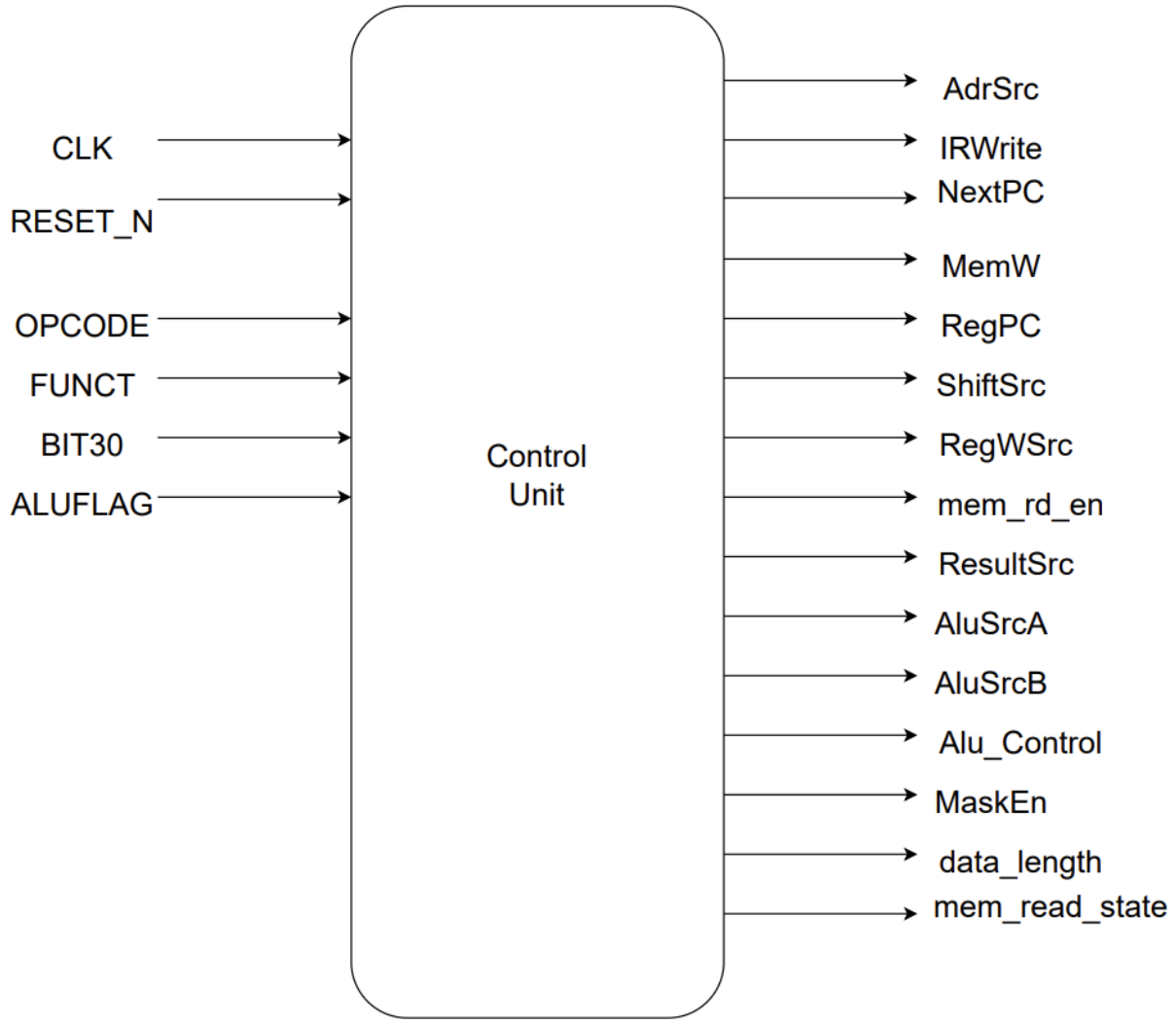
Global CPI is 4

Instruction	CPI (cycle)
I type	4
I type (load)	5
J type	4
R type	4
S type	4
U type	3
B type	4

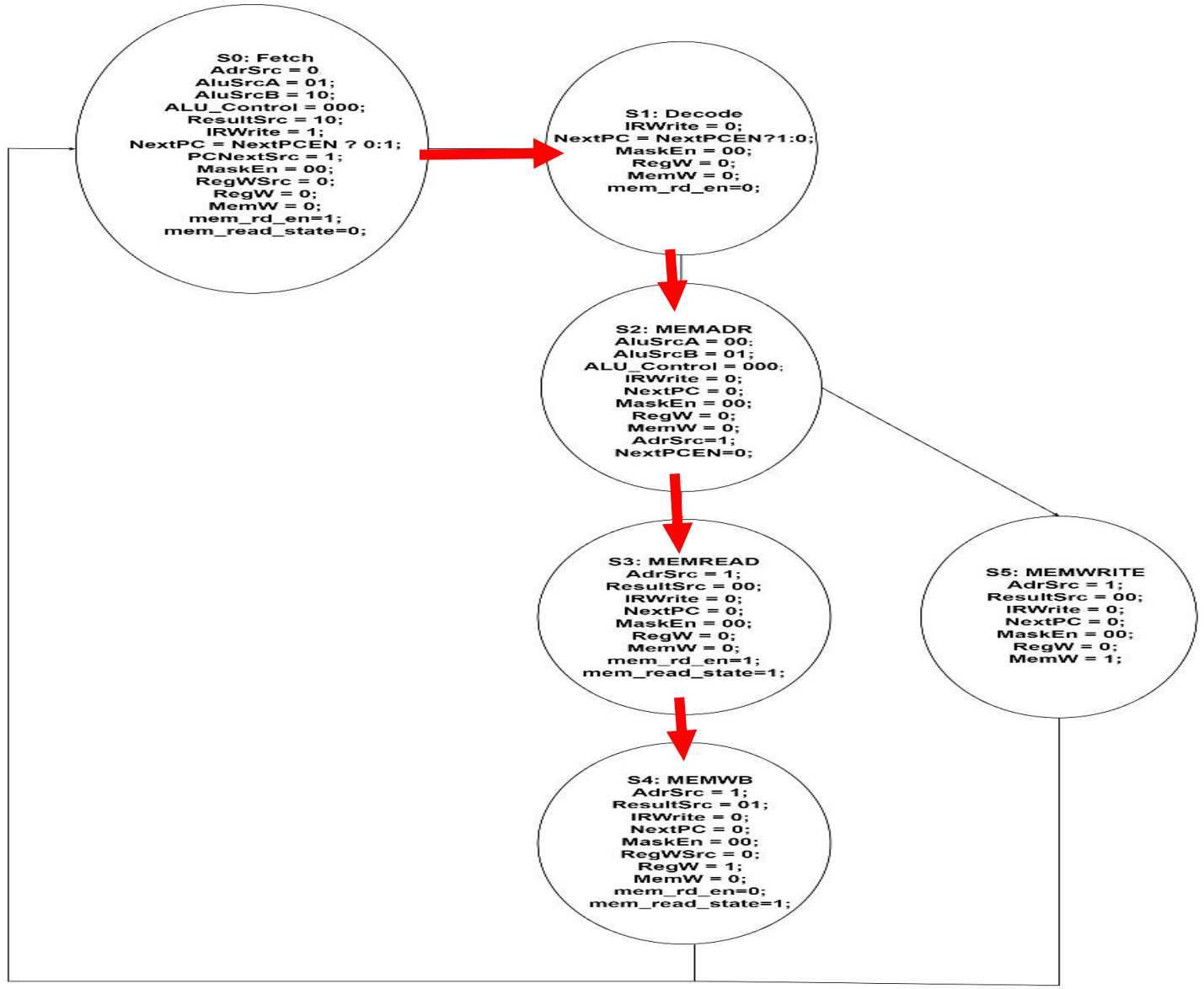
ALU

ALU CONTROL	Operation
000	Addition
001	Subtraction
010	Bitwise And
011	Bitwise OR
100	Bitwise XOR
101	Logical Right Shift
110	Arithmetic Right Shift
111	Logical Left Shift

CONTROL UNIT

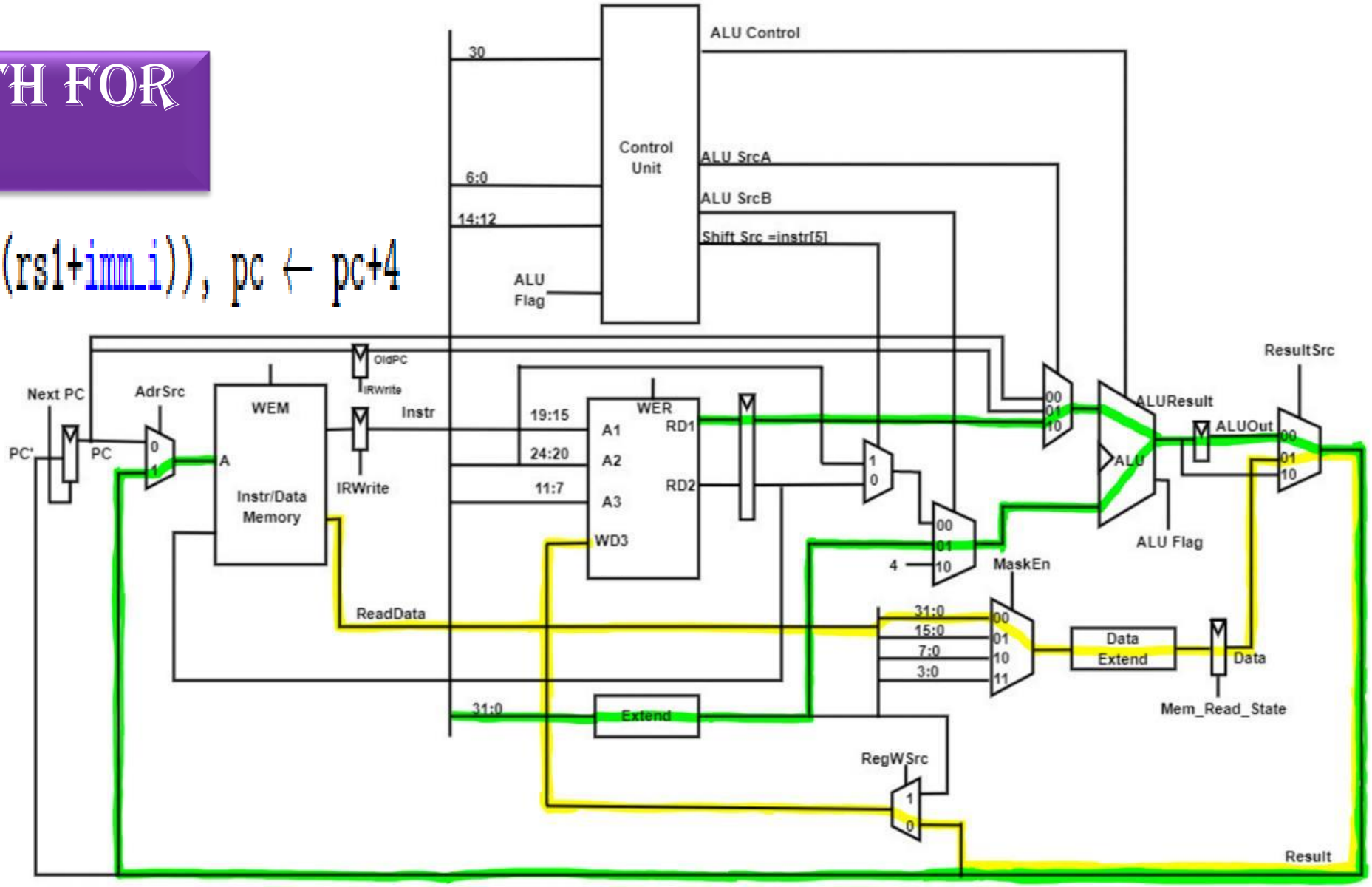


FSM FOR LW (I TYPE)



DATAPATH FOR LW

rd ← sx(m32(rs1+imm_i)), pc ← pc+4



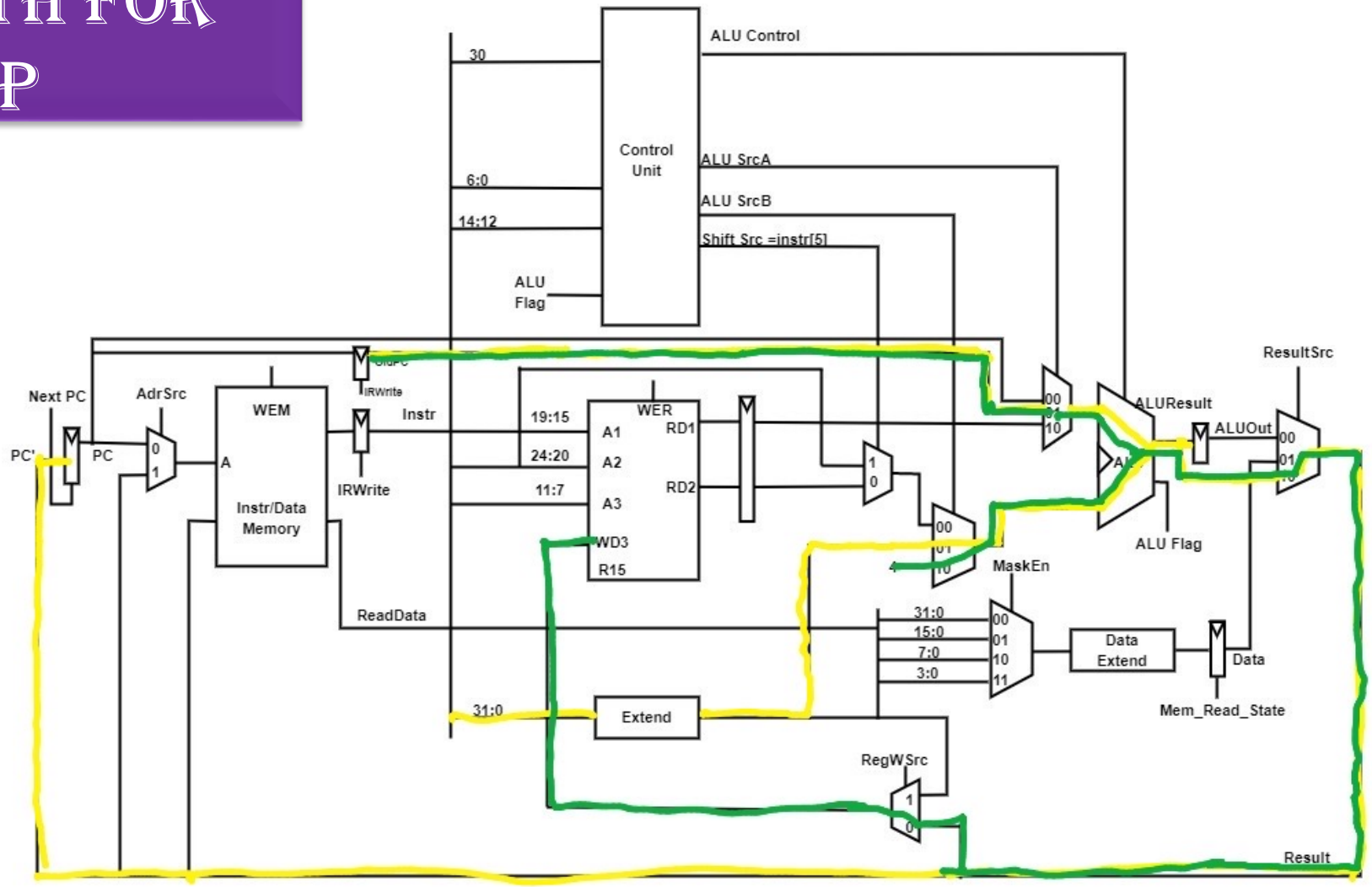
INSTRUCTION – LW


```
lw      a5,-24(s0)
```

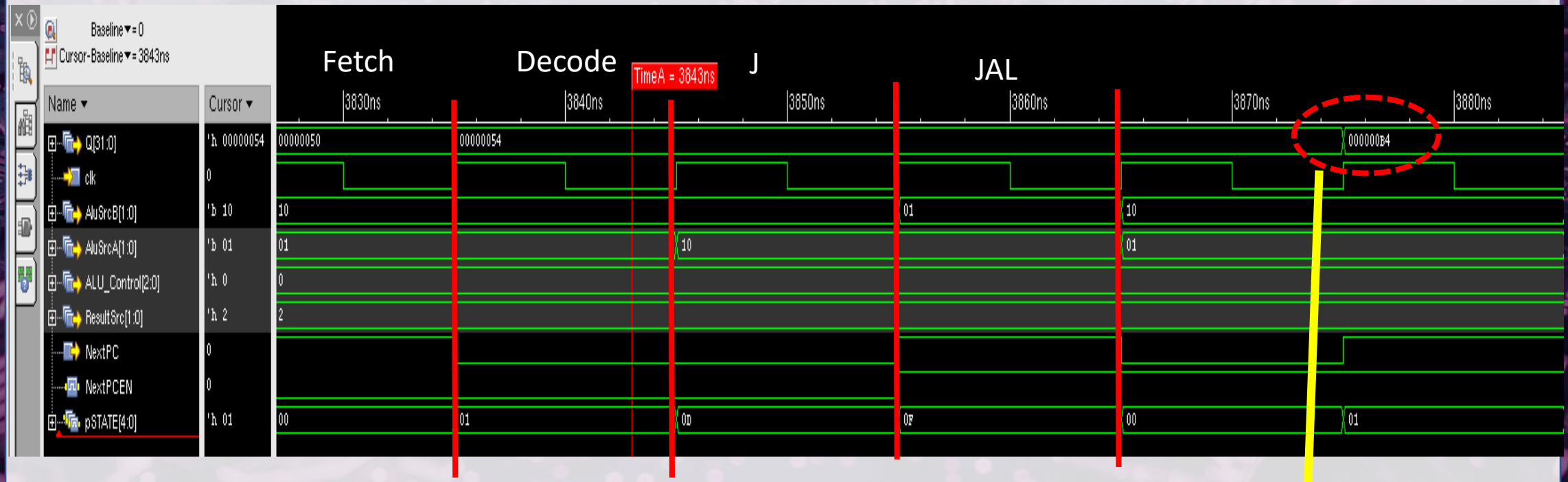
Value written in this clk

FSM FOR JUMP (J TYPE)

DATAPATH FOR JUMP



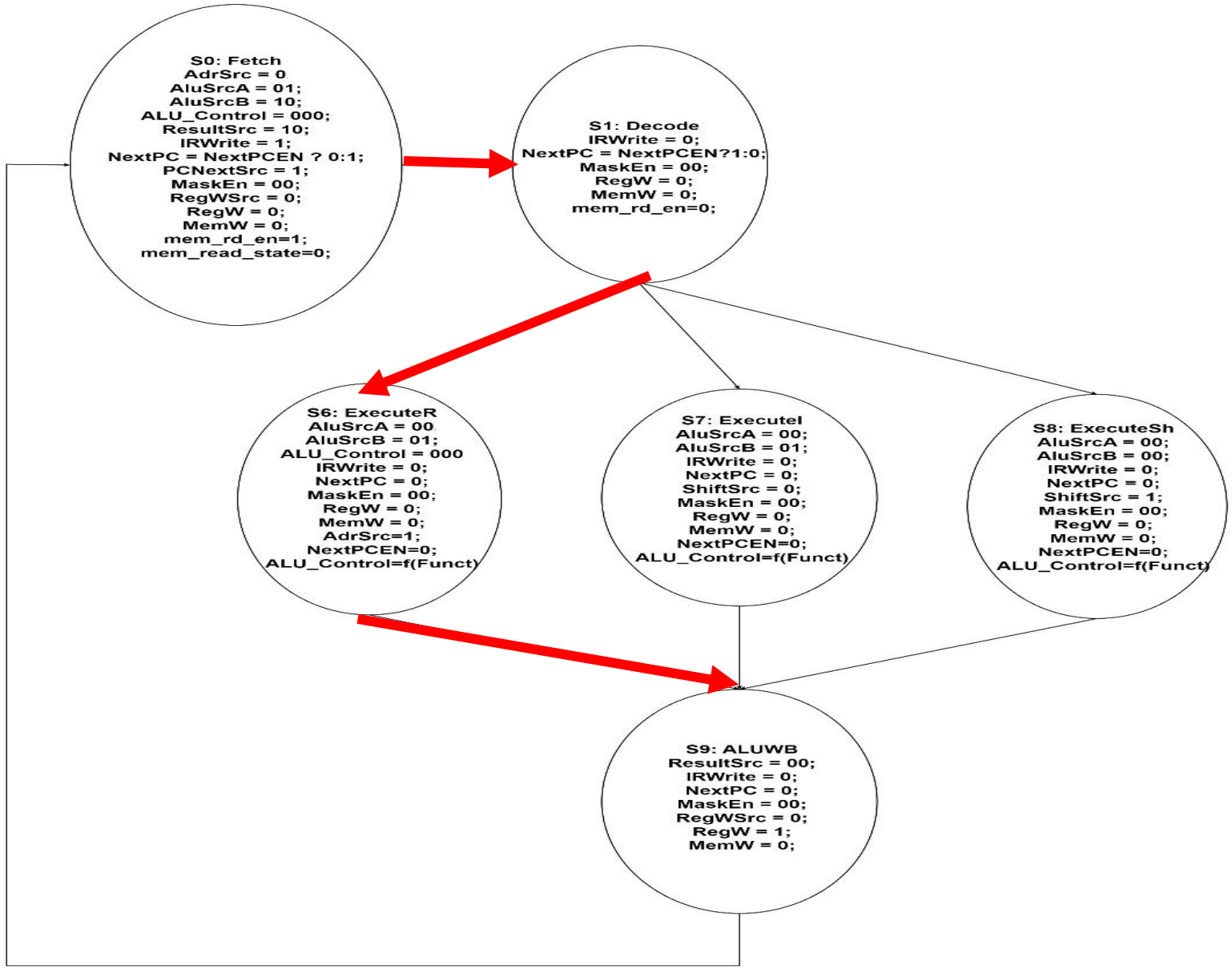
INSTRUCTION – JUMP




```
j       b4 <prime_number+0x9c>
```

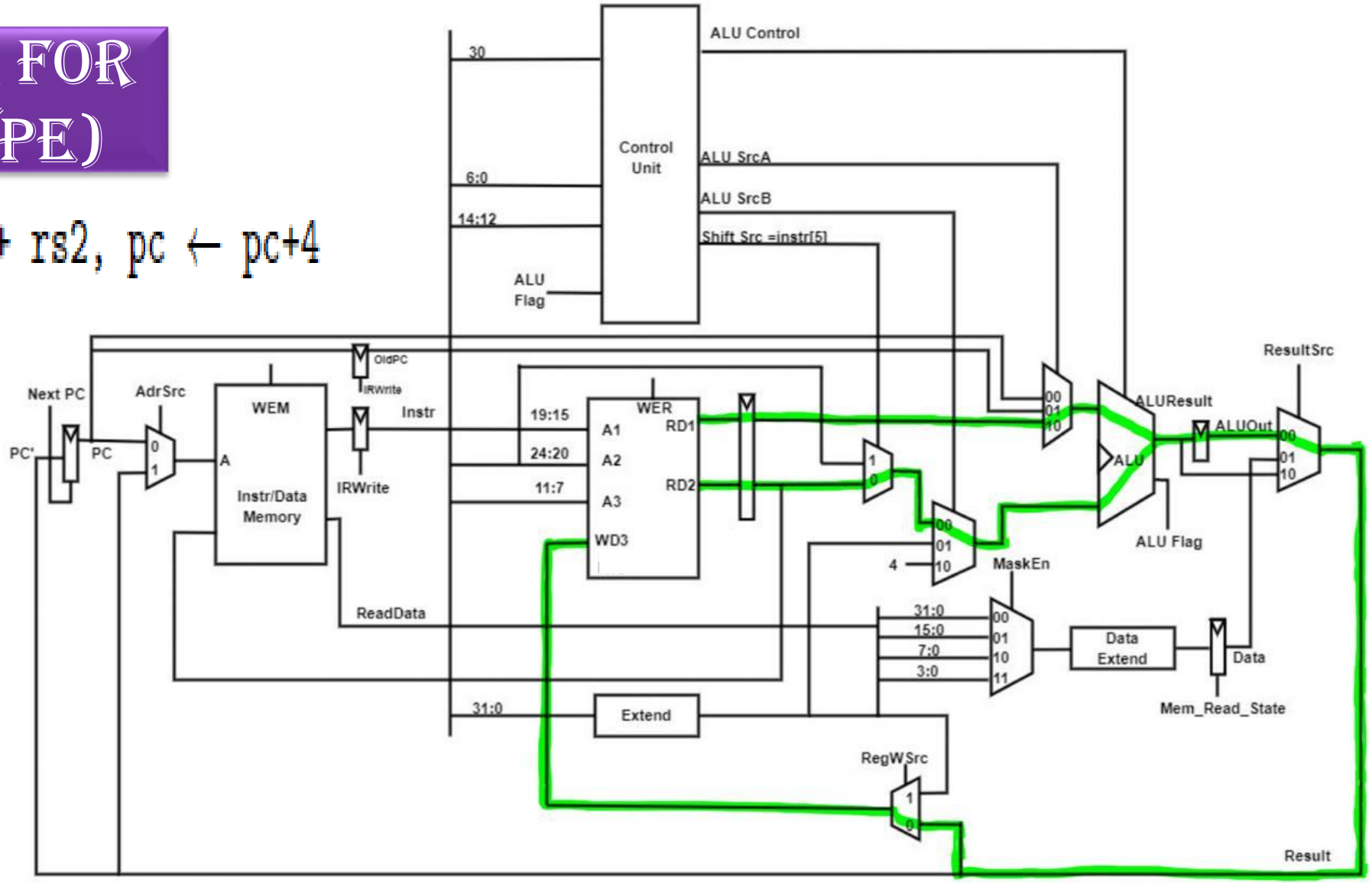
PC went to B4

FSM FOR ADD (R TYPE)

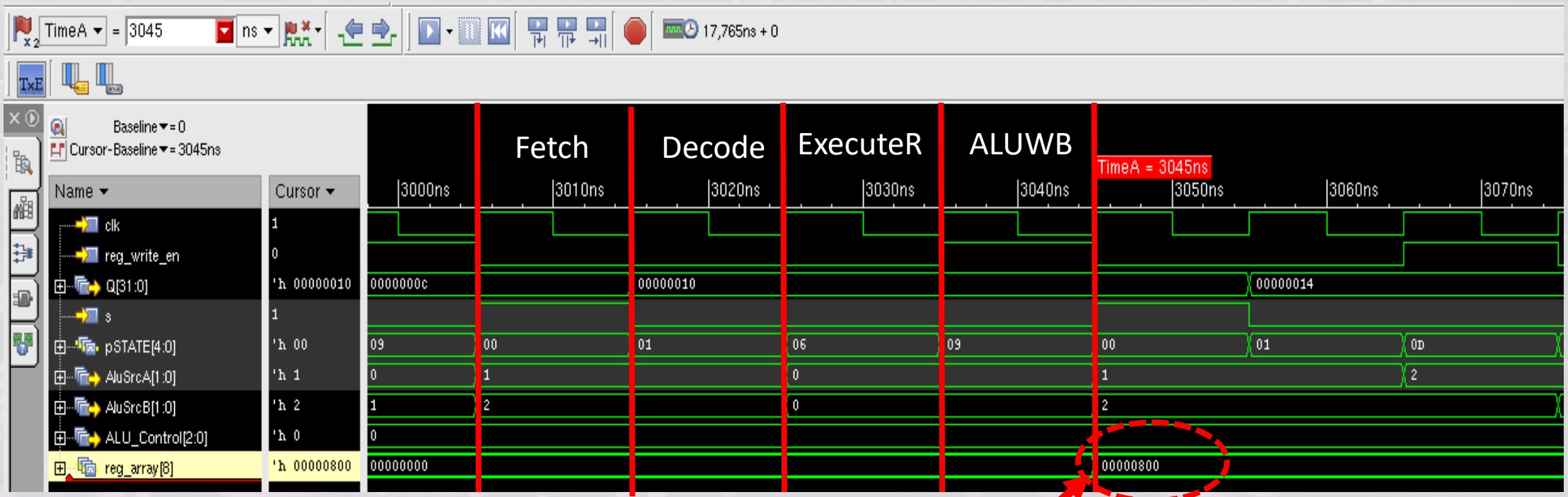


DATAPATH FOR ADD (R TYPE)

rd ← rs1 + rs2, pc ← pc+4



INSTRUCTION – ADD(R TYPE)

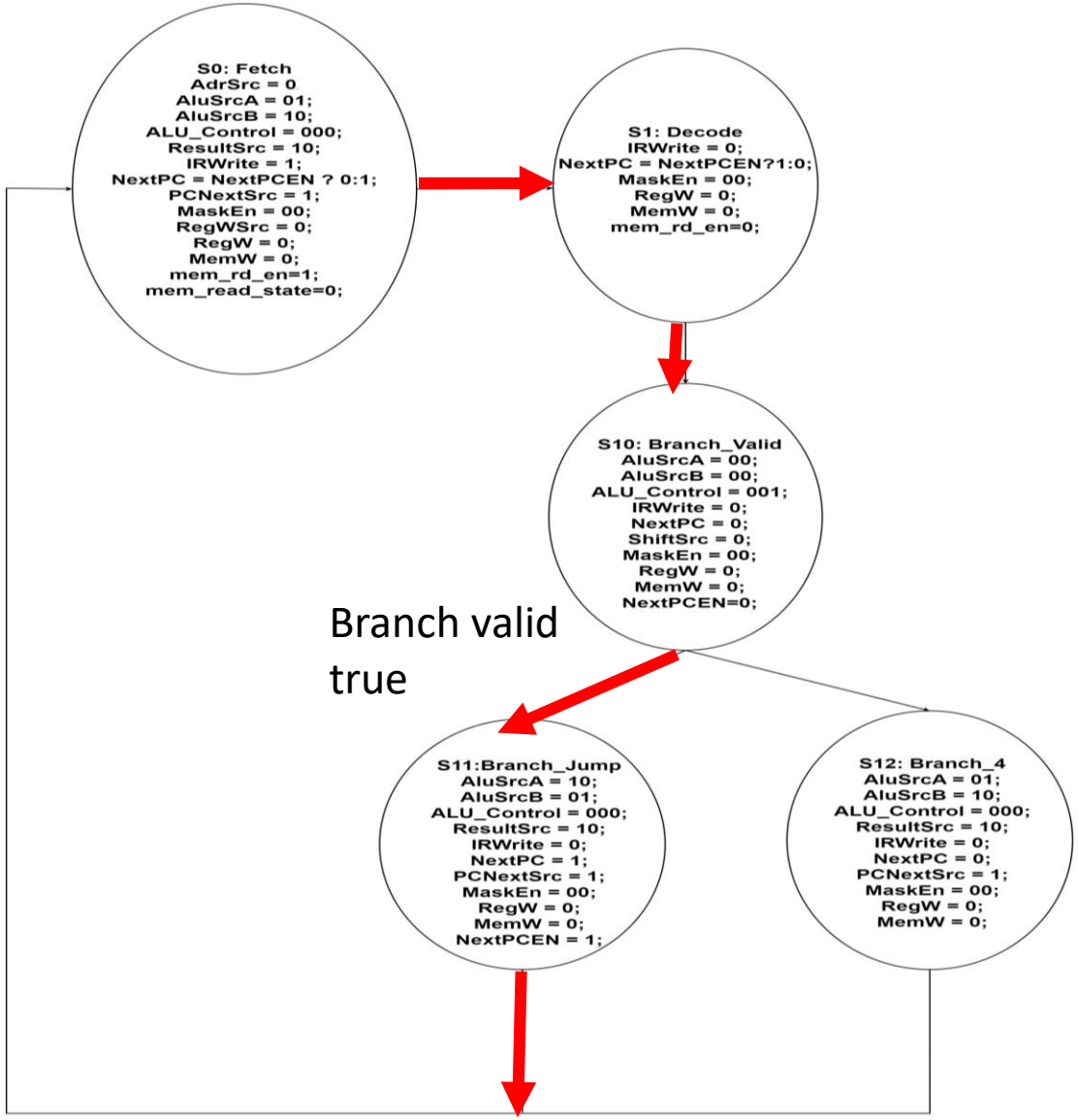



```
10:  00010433          add    s0,sp,zero
```

SP = R2

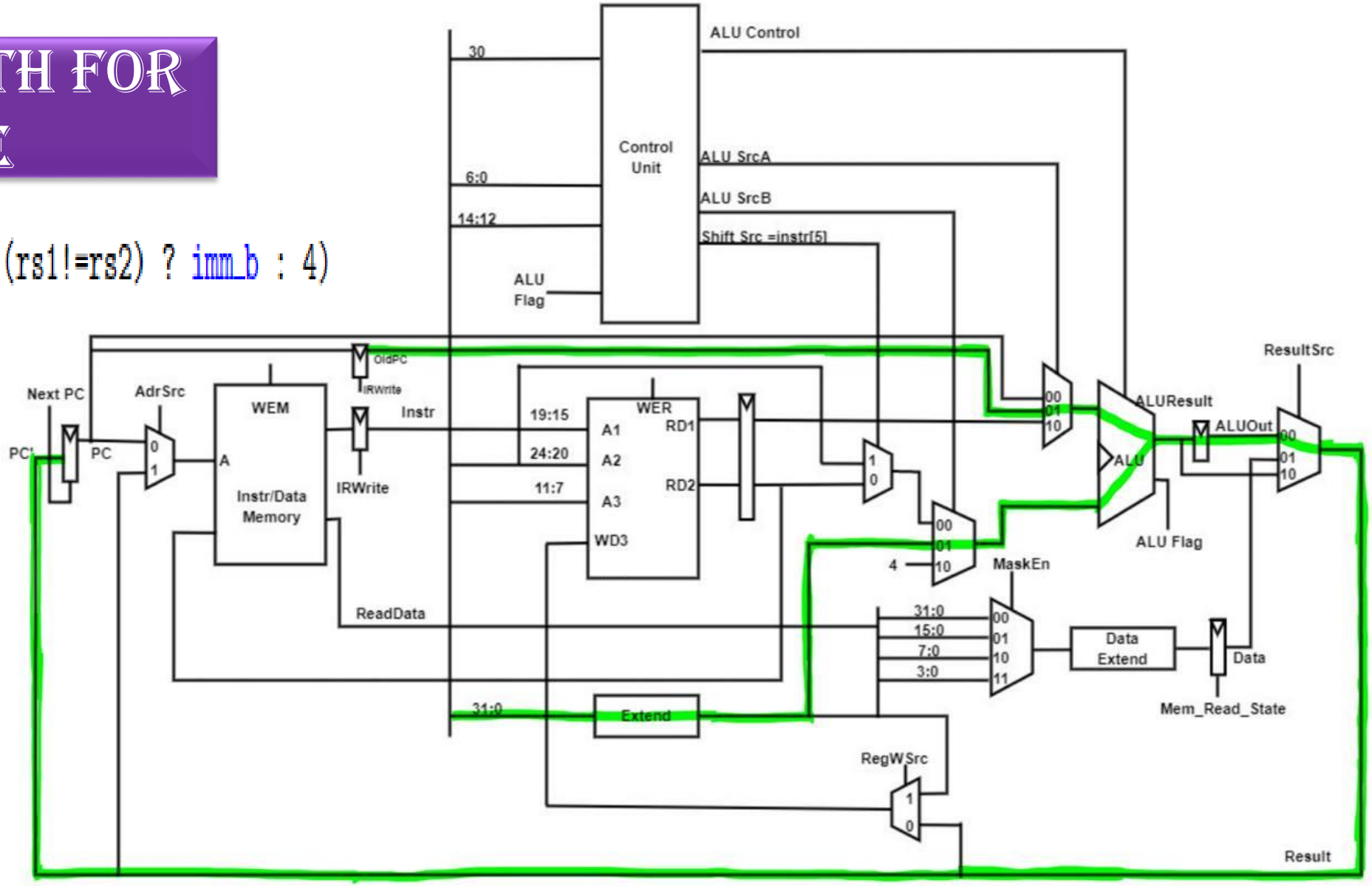
S0 =R8

FSM FOR BNE (B TYPE)

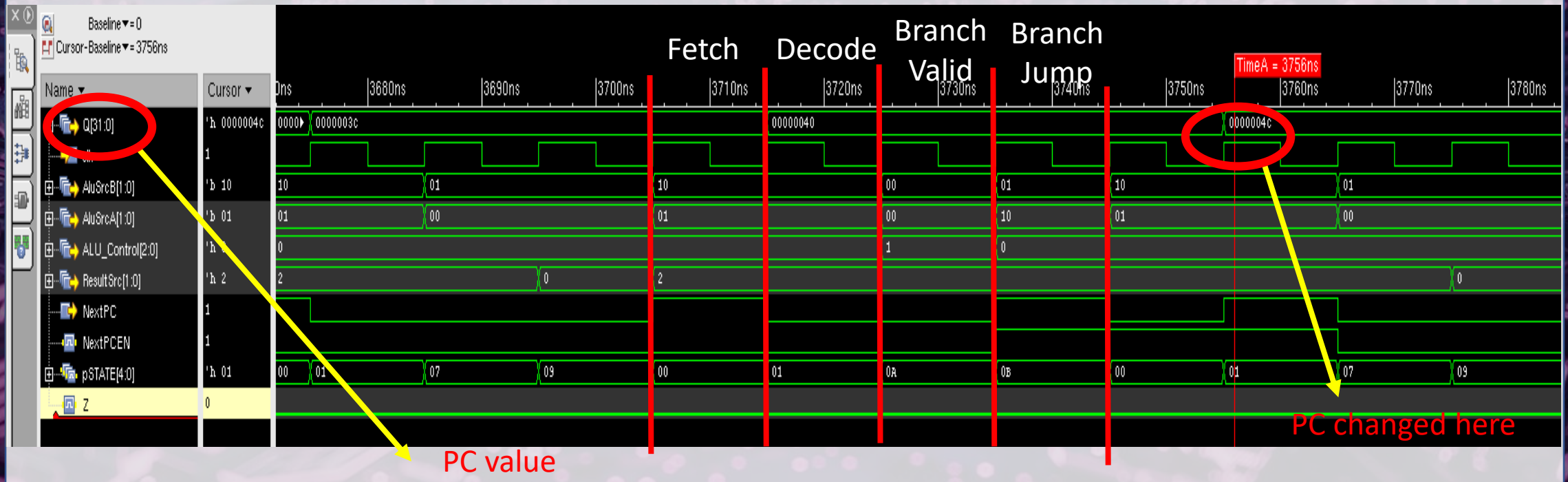


DATAPATH FOR BNE

pc ← pc + ((rs1!=rs2) ? imm_b : 4)

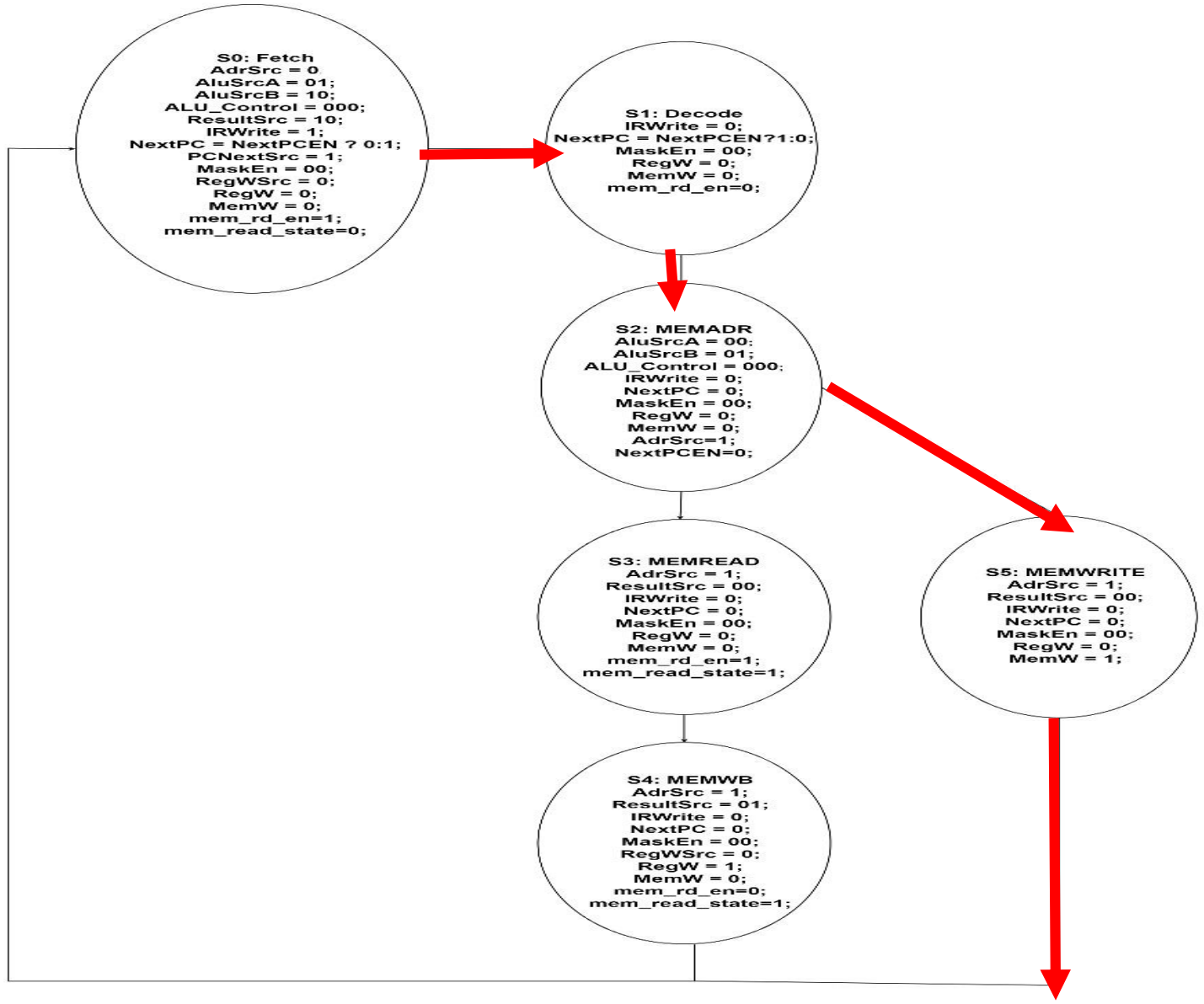


INSTRUCTION – BNE




```
bne    a4,a5,4c <prime_number+0x34>
```

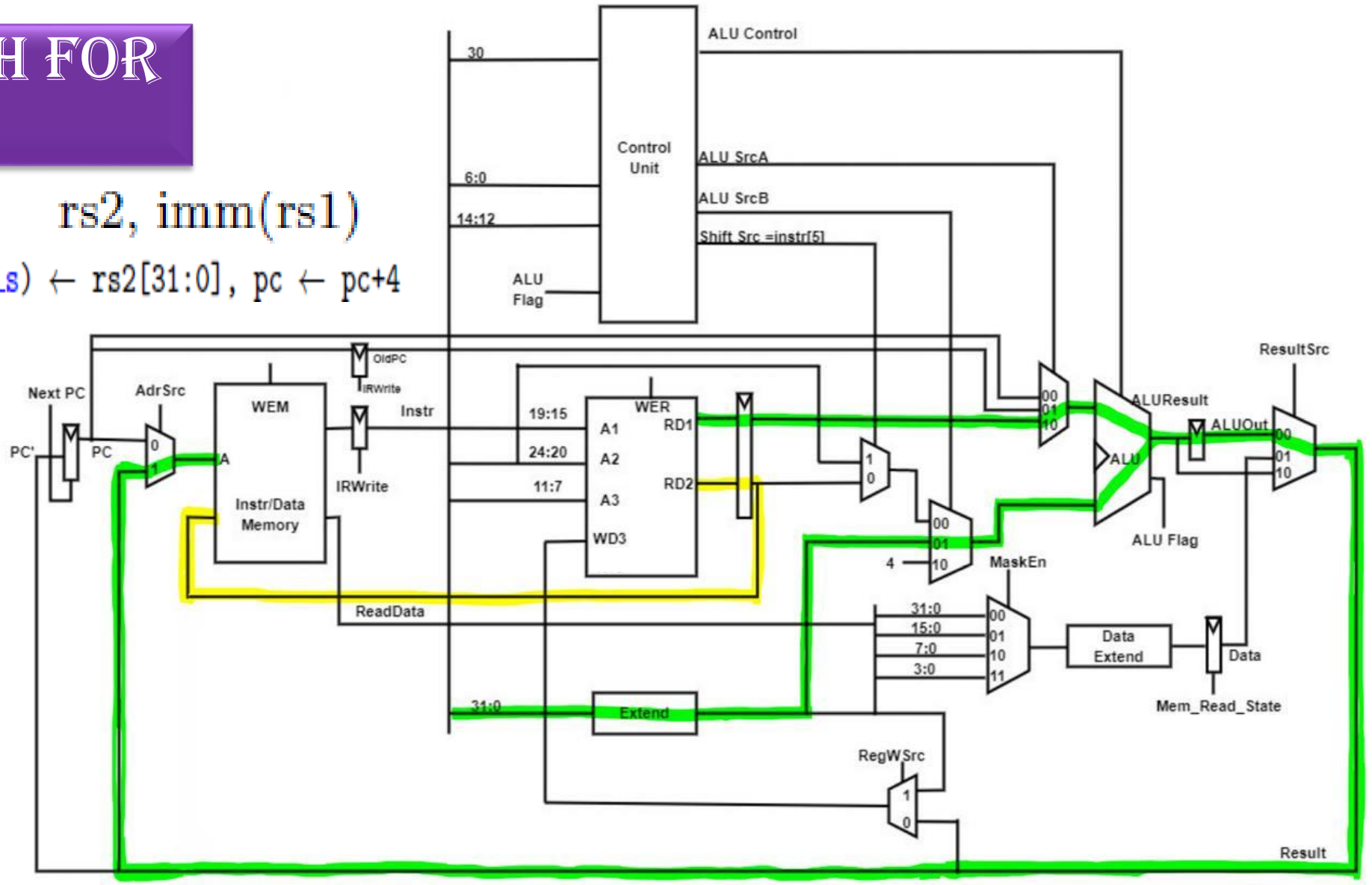
FSM FOR SW (S TYPE)



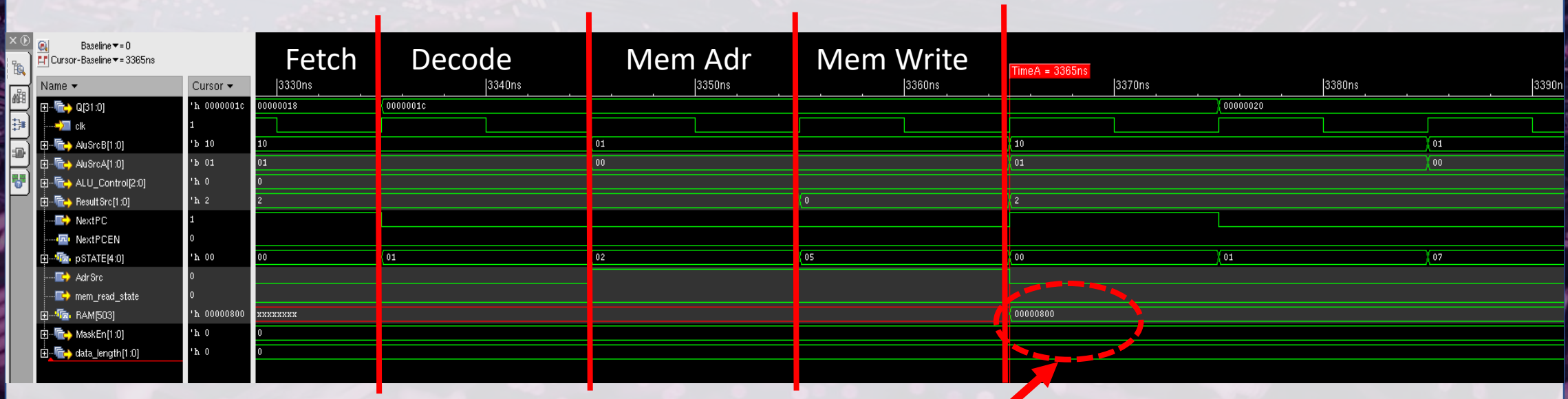
DATAPATH FOR SW

sw rs2, imm(rs1)

m32(rs1+imm_s) ← rs2[31:0], pc ← pc+4



INSTRUCTION – SW

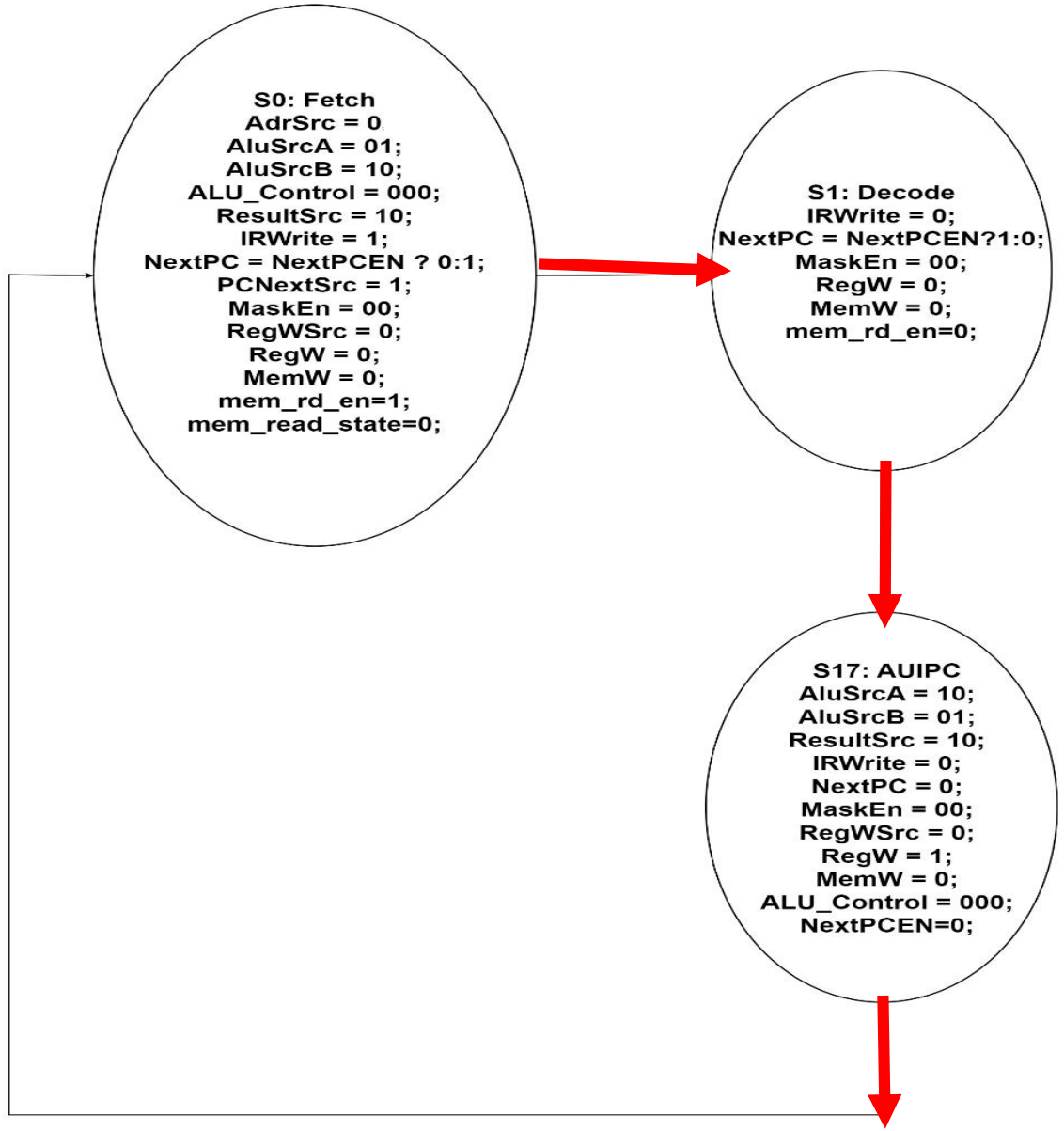



```
sw      s0,28(sp)
```

Value is available in memory from this cycle

- SP means R2
- S0 means R8
- Mem[R2+28] ← R8

FSM FOR AIUPC (U TYPE)

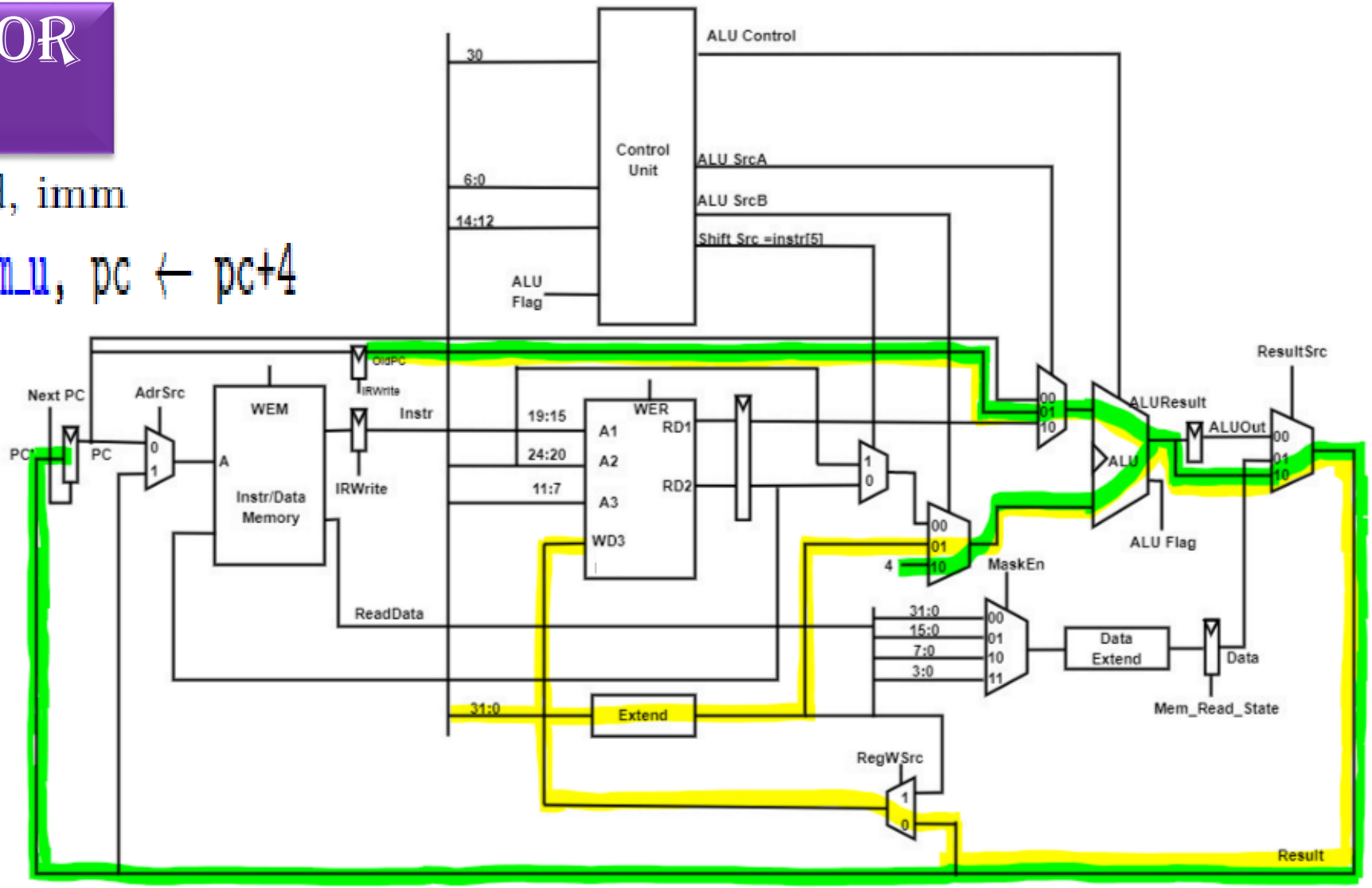


DATAPATH FOR AUIPC

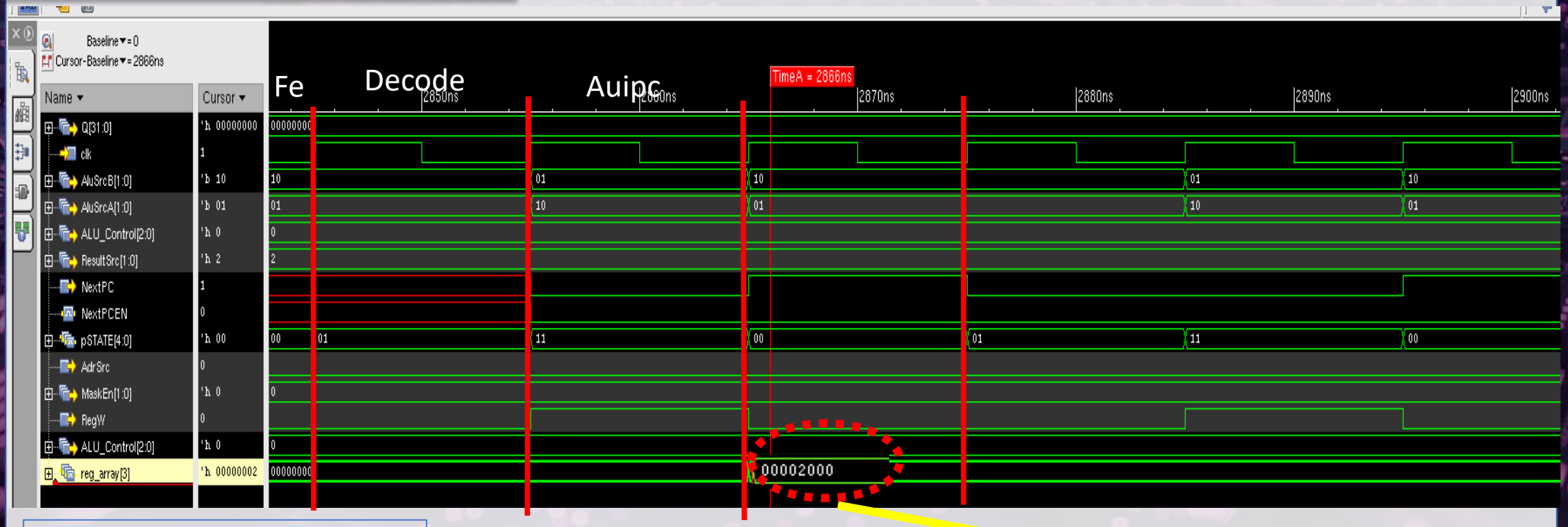
auipc rd, imm

rd ← pc + imm_u, pc ← pc+4

Old PC



INSTRUCTION – AUIPC



Global Pointer =R3

R3 <= PC + immediate

```
auipc  gp,0x2
```

2000(pc+imm_u) is stored in Global pointer

RESULT AND DISCUSSIONS

INSTRUCTIONS COVERED

2592 RV32I Base Instruction Set Encoding [1, p. 104]

31 25	24 20	19 15	14 12	11 7	6 0			
imm[31:12]				rd	0 1 1 0 1 1 1	U-type	lui	rd,imm
imm[31:12]				rd	0 0 1 0 1 1 1	U-type	auipc	rd,imm
imm[20|10:1|11|19:12]				rd	1 1 0 1 1 1 1	J-type	jal	rd,pcrel_21
imm[11:0]		rs1	0 0 0	rd	1 1 0 0 1 1 1	I-type	jalr	rd,imm(rs1)
imm[12|10:5]	rs2	rs1	0 0 0	imm[4:1|11]	1 1 0 0 0 1 1	B-type	beq	rs1,rs2,pcrel_13
imm[12|10:5]	rs2	rs1	0 0 1	imm[4:1|11]	1 1 0 0 0 1 1	B-type	bne	rs1,rs2,pcrel_13
imm[12|10:5]	rs2	rs1	1 0 0	imm[4:1|11]	1 1 0 0 0 1 1	B-type	blt	rs1,rs2,pcrel_13
imm[12|10:5]	rs2	rs1	1 0 1	imm[4:1|11]	1 1 0 0 0 1 1	B-type	bge	rs1,rs2,pcrel_13
imm[12|10:5]	rs2	rs1	1 1 0	imm[4:1|11]	1 1 0 0 0 1 1	B-type	bltu	rs1,rs2,pcrel_13
imm[12|10:5]	rs2	rs1	1 1 1	imm[4:1|11]	1 1 0 0 0 1 1	B-type	bgeu	rs1,rs2,pcrel_13
imm[11:0]		rs1	0 0 0	rd	0 0 0 0 0 1 1	I-type	lb	rd,imm(rs1)
imm[11:0]		rs1	0 0 1	rd	0 0 0 0 0 1 1	I-type	lh	rd,imm(rs1)
imm[11:0]		rs1	0 1 0	rd	0 0 0 0 0 1 1	I-type	lw	rd,imm(rs1)
imm[11:0]		rs1	1 0 0	rd	0 0 0 0 0 1 1	I-type	lbu	rd,imm(rs1)
imm[11:0]		rs1	1 0 1	rd	0 0 0 0 0 1 1	I-type	lhu	rd,imm(rs1)
imm[11:5]	rs2	rs1	0 0 0	imm[4:0]	0 1 0 0 0 1 1	S-type	sb	rs2,imm(rs1)
imm[11:5]	rs2	rs1	0 0 1	imm[4:0]	0 1 0 0 0 1 1	S-type	sh	rs2,imm(rs1)
imm[11:5]	rs2	rs1	0 1 0	imm[4:0]	0 1 0 0 0 1 1	S-type	sw	rs2,imm(rs1)
imm[11:0]		rs1	0 0 0	rd	0 0 1 0 0 1 1	I-type	addi	rd,rs1,imm
imm[11:0]		rs1	0 1 0	rd	0 0 1 0 0 1 1	I-type	slti	rd,rs1,imm
imm[11:0]		rs1	0 1 1	rd	0 0 1 0 0 1 1	I-type	sltiu	rd,rs1,imm
imm[11:0]		rs1	1 0 0	rd	0 0 1 0 0 1 1	I-type	xori	rd,rs1,imm
imm[11:0]		rs1	1 1 0	rd	0 0 1 0 0 1 1	I-type	ori	rd,rs1,imm
imm[11:0]		rs1	1 1 1	rd	0 0 1 0 0 1 1	I-type	andi	rd,rs1,imm
0 0 0 0 0 0 0	shamt	rs1	0 0 1	rd	0 0 1 0 0 1 1	I-type	slli	rd,rs1,shamt
0 0 0 0 0 0 0	shamt	rs1	1 0 1	rd	0 0 1 0 0 1 1	I-type	srli	rd,rs1,shamt
0 1 0 0 0 0 0	shamt	rs1	1 0 1	rd	0 0 1 0 0 1 1	I-type	srai	rd,rs1,shamt
0 0 0 0 0 0 0	rs2	rs1	0 0 0	rd	0 1 1 0 0 1 1	R-type	add	rd,rs1,rs2
0 1 0 0 0 0 0	rs2	rs1	0 0 0	rd	0 1 1 0 0 1 1	R-type	sub	rd,rs1,rs2
0 0 0 0 0 0 0	rs2	rs1	0 0 1	rd	0 1 1 0 0 1 1	R-type	sll	rd,rs1,rs2
0 0 0 0 0 0 0	rs2	rs1	0 1 0	rd	0 1 1 0 0 1 1	R-type	slt	rd,rs1,rs2
0 0 0 0 0 0 0	rs2	rs1	0 1 1	rd	0 1 1 0 0 1 1	R-type	sltu	rd,rs1,rs2
0 0 0 0 0 0 0	rs2	rs1	1 0 0	rd	0 1 1 0 0 1 1	R-type	xor	rd,rs1,rs2
0 0 0 0 0 0 0	rs2	rs1	1 0 1	rd	0 1 1 0 0 1 1	R-type	srl	rd,rs1,rs2
0 1 0 0 0 0 0	rs2	rs1	1 0 1	rd	0 1 1 0 0 1 1	R-type	sra	rd,rs1,rs2
0 0 0 0 0 0 0	rs2	rs1	1 1 0	rd	0 1 1 0 0 1 1	R-type	or	rd,rs1,rs2
0 0 0 0 0 0 0	rs2	rs1	1 1 1	rd	0 1 1 0 0 1 1	R-type	and	rd,rs1,rs2

CHALLENGES

- Handling PC Logic- Adding old pc register to our existing data path.
- Handling Instruction and data conflict- We added two separate register in the datapath.

FUTURE IMPROVEMENT

- WE can implement these special type of instructions
- We can use pipeline architecture to minimize run time.

0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0	0 0 0	0 0 0 0 0	1 1 1 0 0 1 1		ecall
0 0 0 0 0 0 0 0 0 0 0 1	0 0 0 0 0	0 0 0	0 0 0 0 0	1 1 1 0 0 1 1		ebreak
csr[11:0]	rs1	0 0 1	rd	1 1 1 0 0 1 1	I-type	csrrw rd,csr,rs1
csr[11:0]	rs1	0 1 0	rd	1 1 1 0 0 1 1	I-type	csrrs rd,csr,rs1
csr[11:0]	rs1	0 1 1	rd	1 1 1 0 0 1 1	I-type	csrrc rd,csr,rs1
csr[11:0]	zimm[4:0]	1 0 1	rd	1 1 1 0 0 1 1	I-type	csrrwi rd,csr,zimm
csr[11:0]	zimm[4:0]	1 1 0	rd	1 1 1 0 0 1 1	I-type	csrrsi rd,csr,zimm
csr[11:0]	zimm[4:0]	1 1 1	rd	1 1 1 0 0 1 1	I-type	csrrci rd,csr,zimm

Thank You!

Dr. A.B.M. Harun-ur-Rashid
Professor
Electrical & Electronic Engeineering,
BUET

NEURAL SEMICONDUCTOR

Nafis Sadik
Lecturer
Electrical & Electronic
Engineering,
BUET

Ashiqur Rasul
Lecturer
Electrical
& Electronic
Engineering,
BUET