CS 61C:
Great Ideas in Computer Architecture

Lecture 11: *Datapath*

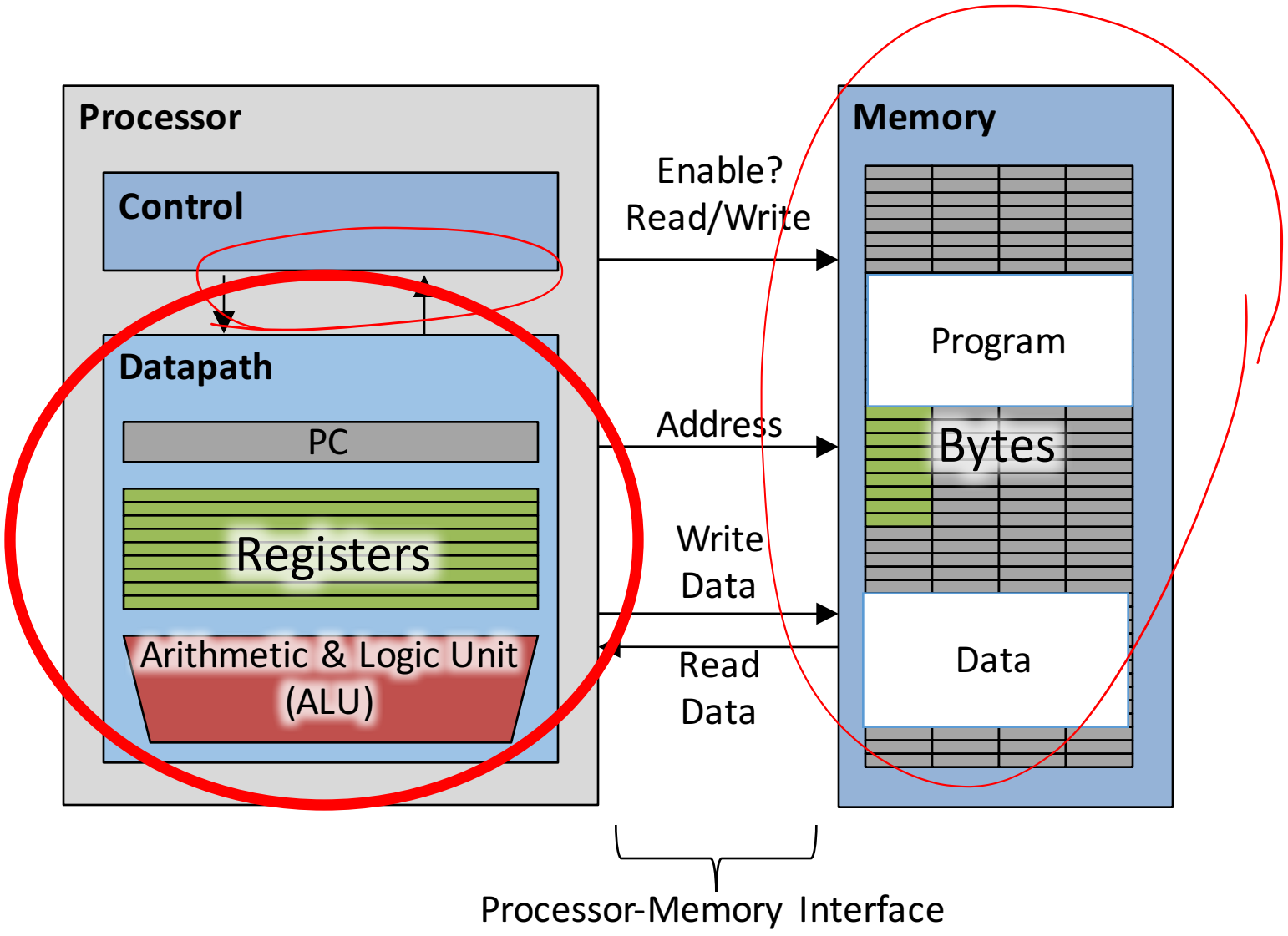
Bernhard Boser & Randy Katz

http://inst.eecs.berkeley.edu/~cs61c

Agenda

- **MIPS Datapath**
  - **add**
    - **instruction**
    - **register transfer level**
    - **circuit**
    - **timing**
    - **control**
  - and, …
  - addi, …
  - beq, …
  - j
  - lw, sw
- And in Conclusion, …

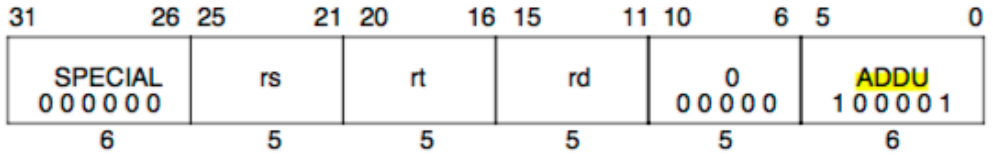
Processor


Processor
Control
Datapath
PC
Registers
Arithmetic & Logic Unit (ALU)
Memory
Program
Bytes
Data
Enable?
Read/Write
Address
Write Data
Read Data
Processor-Memory Interface

Datapath: **addu** - Specs

- Green Card
- MIPS ISA documentation

Add Unsigned Word **ADDU**



Format: ADDU rd, rs, rt MIPS I

Purpose: To add 32-bit integers.

Description: rd ← rs + rt

The 32-bit word value in GPR *rt* is added to the 32-bit value in GPR *rs* and the 32-bit arithmetic result is placed into GPR *rd*.

No Integer Overflow exception occurs under any circumstances.

Restrictions:

On 64-bit processors, if either GPR *rt* or GPR *rs* do not contain sign-extended 32-bit values (bits 63..31 equal), then the result of the operation is undefined.

Operation:

```
if (NotWordValue(GPR[rs]) or NotWordValue(GPR[rt])) then UndefinedResult() endif
temp    ←GPR[rs] + GPR[rt]
GPR[rd]← sign_extend(temp31..0)
```

Exceptions:

None

Programming Notes:

The term "unsigned" in the instruction name is a misnomer; this operation is 32-bit modulo arithmetic that does not trap on overflow. It is appropriate for arithmetic which is not signed, such as address arithmetic, or integer arithmetic environments that ignore overflow, such as "C" language arithmetic.

Datapath Design: **addu**

- Instruction

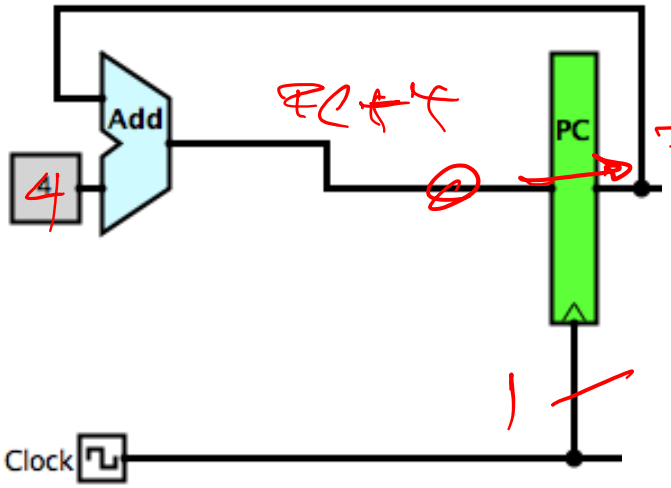
Type	31 …		format (bits)			… 0
R	Op Code (6)	rs (5)	rt (5)	rd (5)	shmt (5)	funct (6)
	0x0	rs	rt	rd	0	0x21

- Effect - "Register transfer level, RTL"
  - **`PC ← PC + 4`**
  - **`R[$rd] ← R[$rs] + R[$rt]`**

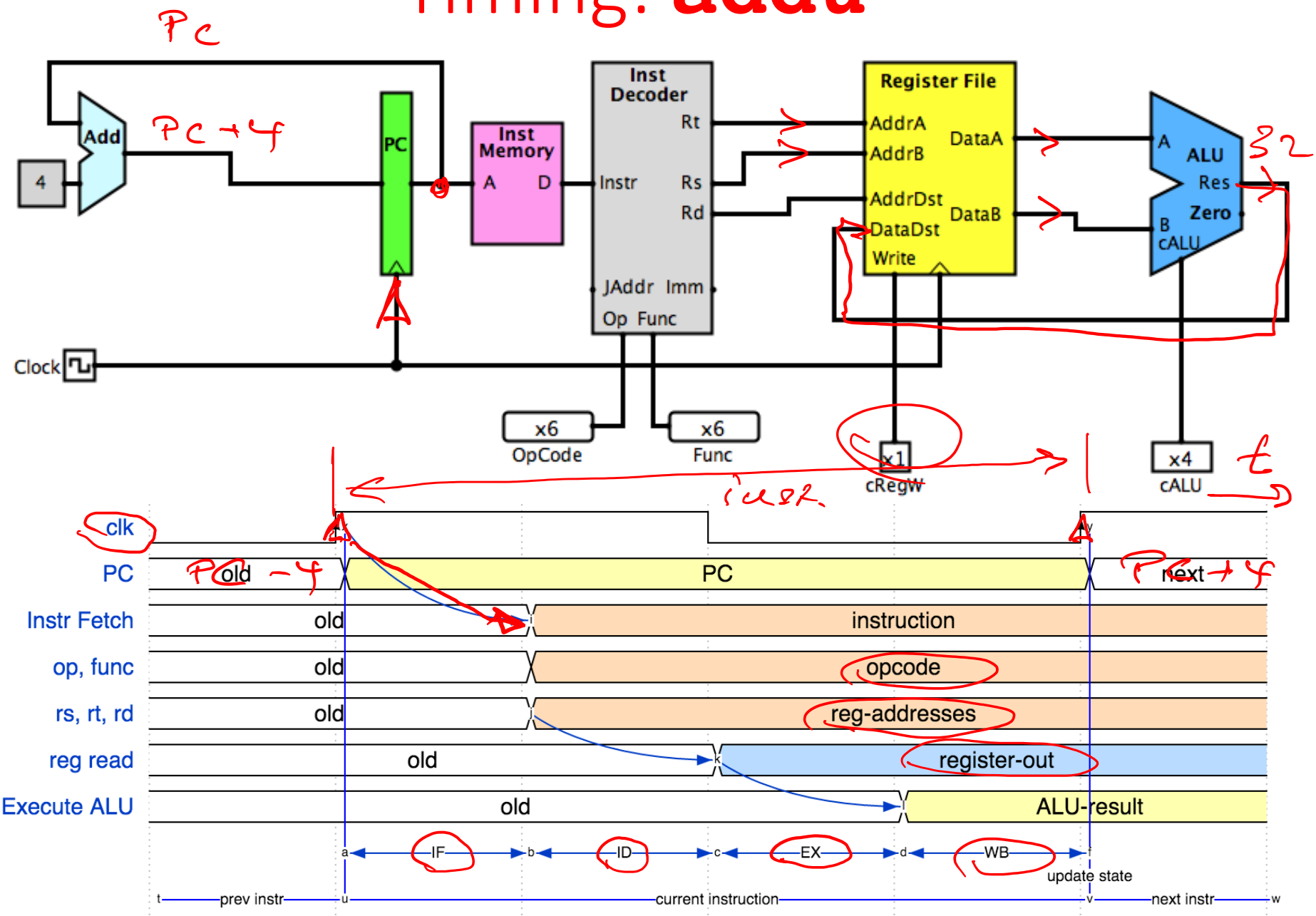
Circuit: **addu**

Type	31 …	format (bits)				… 0
R	Op Code (6)	rs (5)	rt (5)	rd (5)	shmt (5)	funct (6)
	0x0	rs	rt	rd	0	0x21

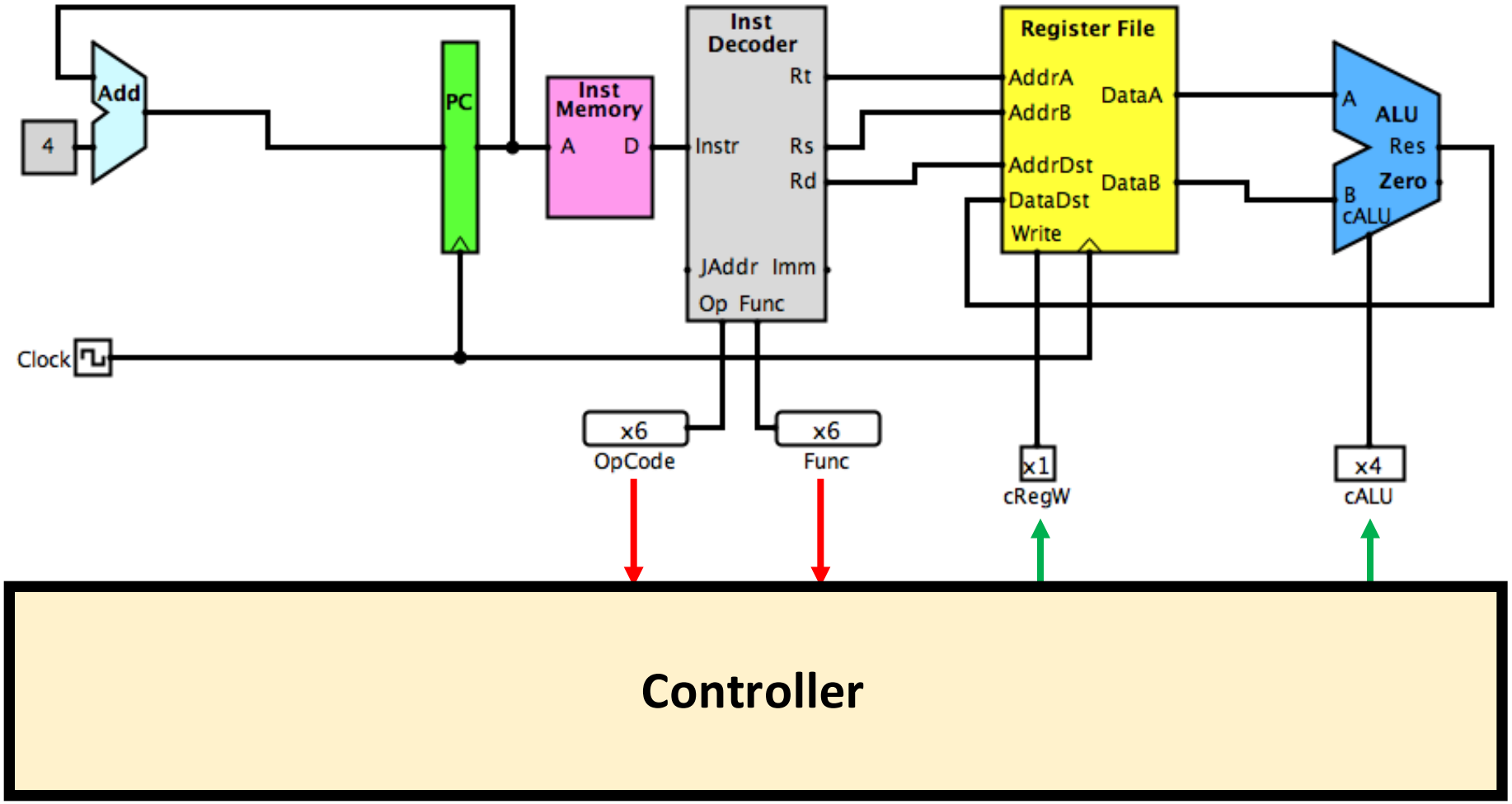
- Register transfer level, RTL
  - `PC ← PC + 4`
  - `R[$rd] ← R[$rs] + R[$rt]`



Timing: **addu**


Add
4
PC
Inst Memory
A
D
Inst Decoder
Rt
Instr
Rs
Rd
JAddr
Imm
Op
Func
Register File
AddrA
AddrB
DataA
AddrDst
DataB
DataDst
Write
A
ALU
Res
Zero
B
cALU
Clock
x6
OpCode
x6
Func
x1
cRegW
x4
cALU
clk
PC
old
PC
next
Instr Fetch
old
instruction
op, func
old
opcode
rs, rt, rd
old
reg-addresses
reg read
old
register-out
Execute ALU
old
ALU-result
a
IF
b
ID
c
EX
d
WB
f
update state
t
prev instr
u
current instruction
v
next instr
w

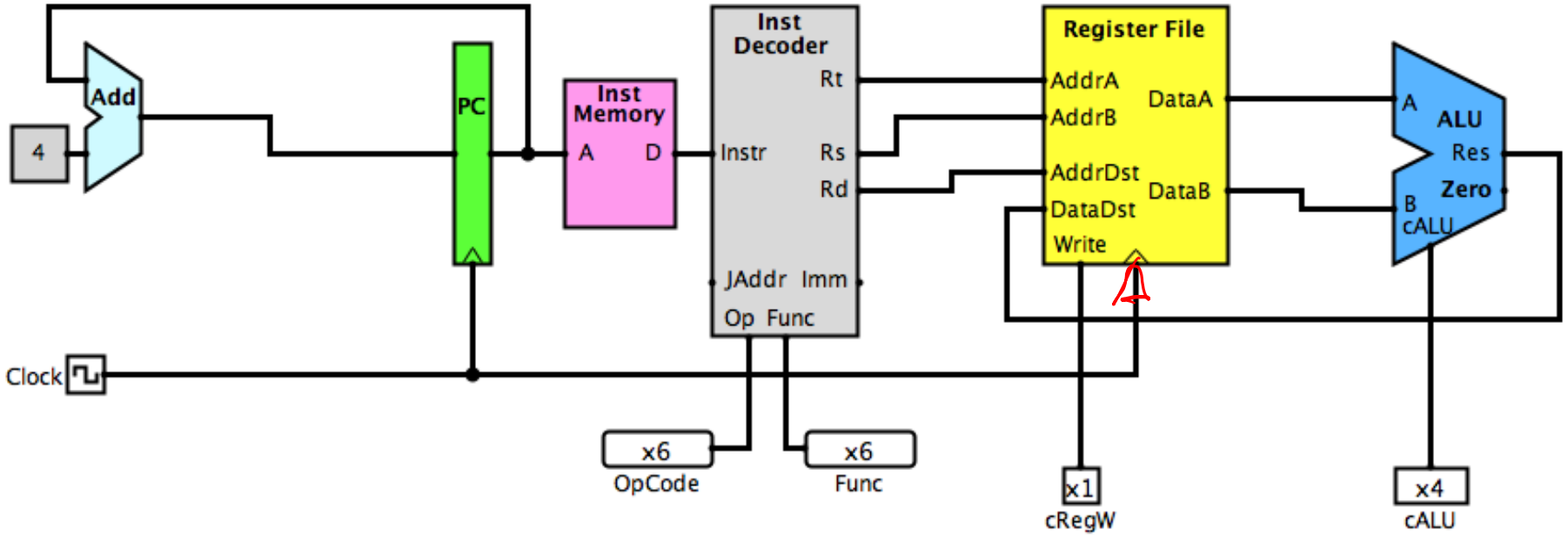
Controller

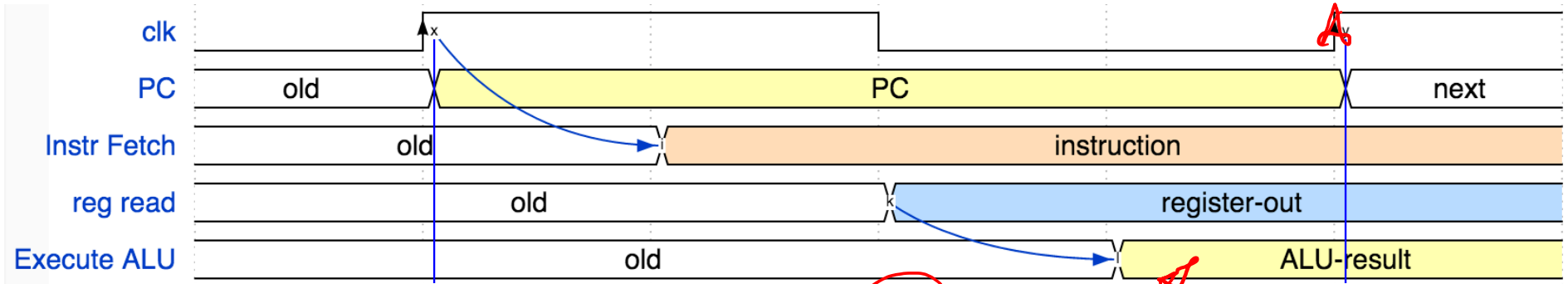

Add
4
PC
Inst Memory
A
D
Inst Decoder
Rt
Instr
Rs
Rd
JAddr
Imm
Op
Func
Register File
AddrA
AddrB
DataA
AddrDst
DataB
DataDst
Write
A
ALU
Res
Zero
B
cALU
Clock
x6
OpCode
x6
Func
x1
cRegW
x4
cALU
Controller

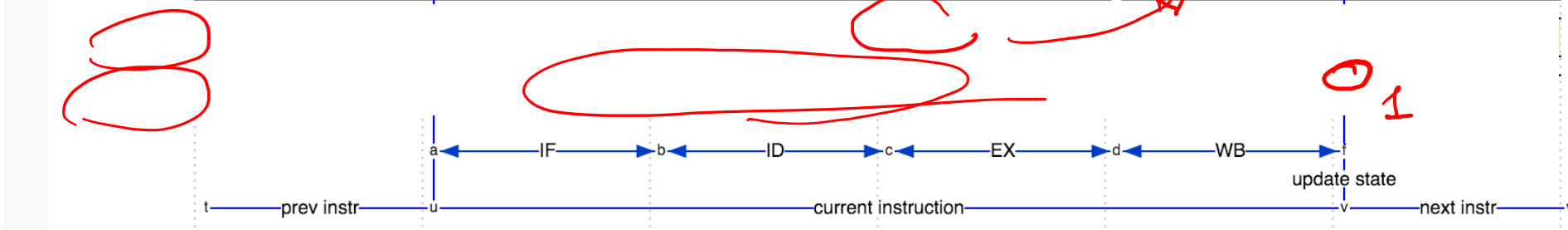
Control Signals

- **cALU**
  - correct ALU operation: add, sub, or, …
- **cRegW**
  - Assert (1) to write to dest register
  - Registers unchanged if 0
- Determined by
  - **OpCode**
  - **Func** code (R-type instructions)

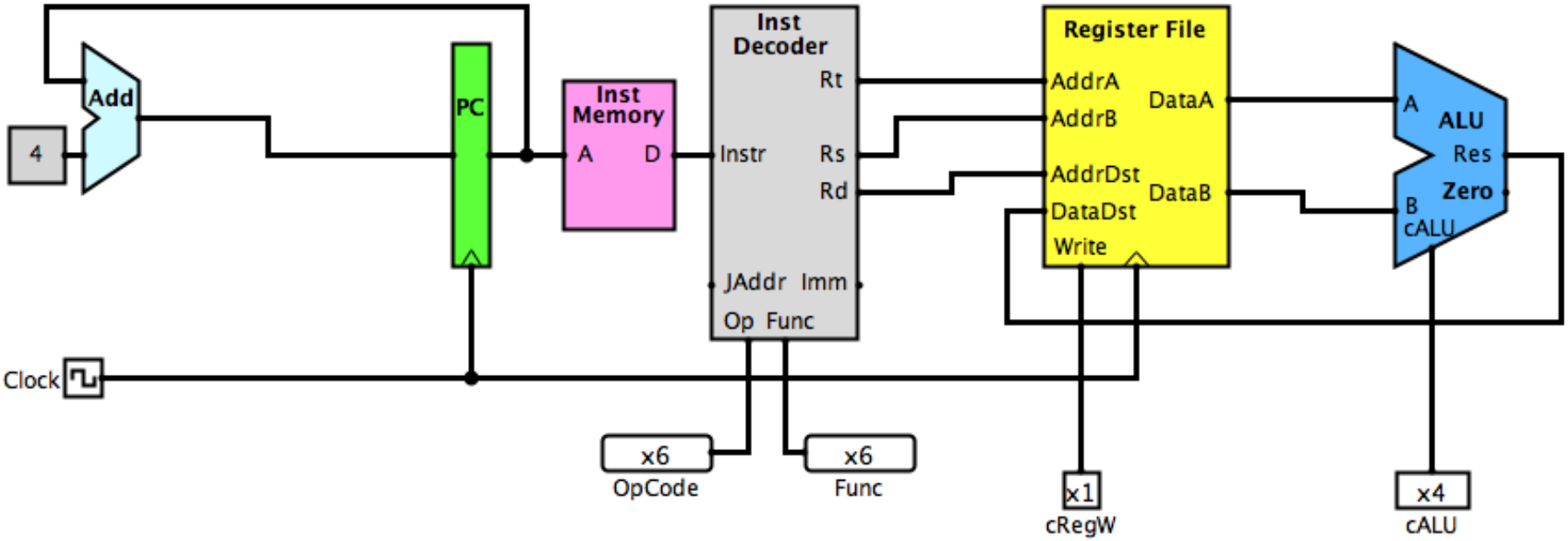
Control: **addu**







Control: **addu**



- Control
  - Determined from **`OpCode`** and **`Func`**

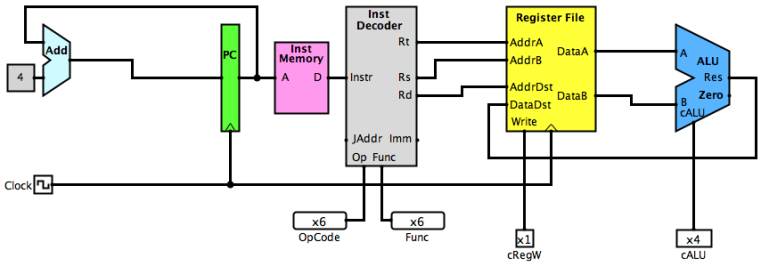
OpCode	Func	cALU	cRegW
0x0	0x21	add	1

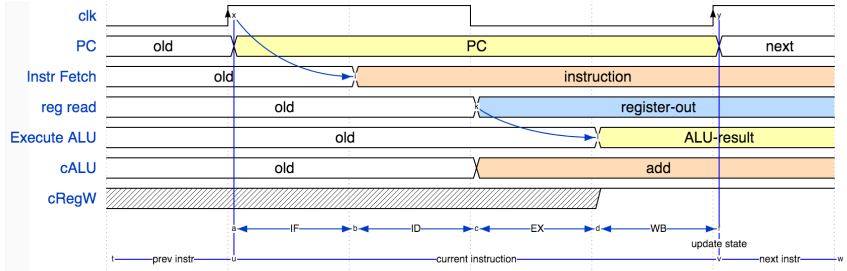
- We will design the controller later
  - For now, let's just remember the required signals

Datapath for **addu** - Summary

Type	31 …	format (bits)				… 0
R	Op Code (6)	rs (5)	rt (5)	rd (5)	shmt (5)	funct (6)
	0x0	rs	rt	rd	0	0x24

- Register transfer level, RTL
  - **`PC ← PC + 4`**
  - **`R[$rd] ← R[$rs] + R[$rt]`**





OpCode	Func	cALU	cRegW
0x0	0x21	add	1

The 61C Mystery Channel Presents …

The mystery of add and addu

Detective 61C checks the evidence

Smoking Gun

aka 1-bit adder truth table

A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Which of the following statements is correct?

Answer	Statement
A	TT is for *signed* addition
B	TT is for *unsigned* addition
C	TT is incorrect, its sole purpose is to hide the crime
D	For 32-bit this is irrelevant, anyway
E	TT is for *signed* and *unsigned* addition, since (Lecture 1) with two's complement representation they are the SAME

Sinister Motives

- What's the purpose of having two instructions, **`add addu`**?
  - Midterm "trick" questions?
- Overflow differs for *signed* and *unsigned*!
  - Lecture 1
- **`add`**
  - Raises exception on overflow, assuming operands are two's complement signed
- **`addu`**
  - Ignores overflow
  - It can therefore be used for *signed* AND *unsigned* two's complement numbers
  - Despite its name!
- What about exceptions on unsigned overflow?
  - Design your own MIPS!
  - Now you know why you need to take 61C!

When use **`add, addu`**?

- Address (pointer) arithmetic:
  - signed overflow makes no sense
  - always use **`addu`**
- Unsigned ints
  - **`addu`**
  - won't report overflow!
- Signed ints
  - **`add`** if you care about overflow
    - e.g. Fortran
  - **`addu`** if you don't care about overflow
    - e.g. C

So …

- Since in 61C we program in C
  - we always use **addu**
  - and never **add**?

And after this interlude …

- 61C returns to
  - rigorous
  - always crystal clear
  - and never confusing
  - engineering facts!

- No more mystery!

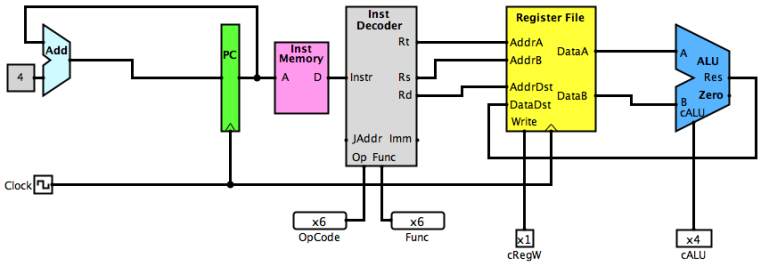
Agenda

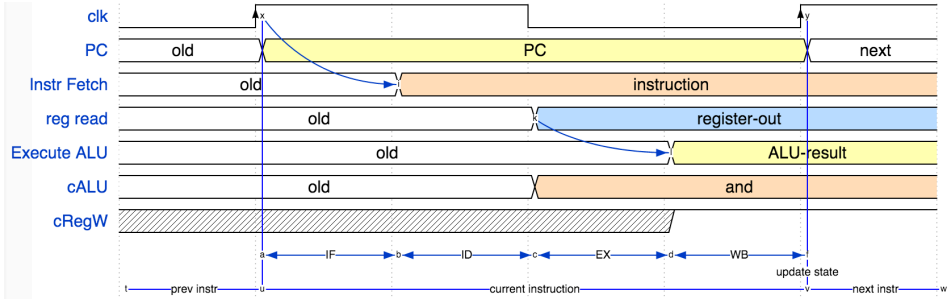
- **MIPS Datapath**
  - add
    - instruction
    - register transfer level
    - circuit
    - timing
    - control
  - **and, …**
  - addi, ...
  - beq, …
  - j
  - lw, sw
- And in Conclusion, ...

Datapath for **and**

Type	31 …	format (bits)				… 0
R	Op Code (6)	rs (5)	rt (5)	rd (5)	shmt (5)	funct (6)
	0x0	rs	rt	rd	0	0x24

- Register transfer level, RTL
  - **`PC ← PC + 4`**
  - **`R[$rd] ← R[$rs] & R[$rt]`**





OpCode	Func	cALU	cRegW
0x0	0x24	and	1

- Same as **`add`** – except cALU
  - Ditto for **`sub, or, xor, …`**

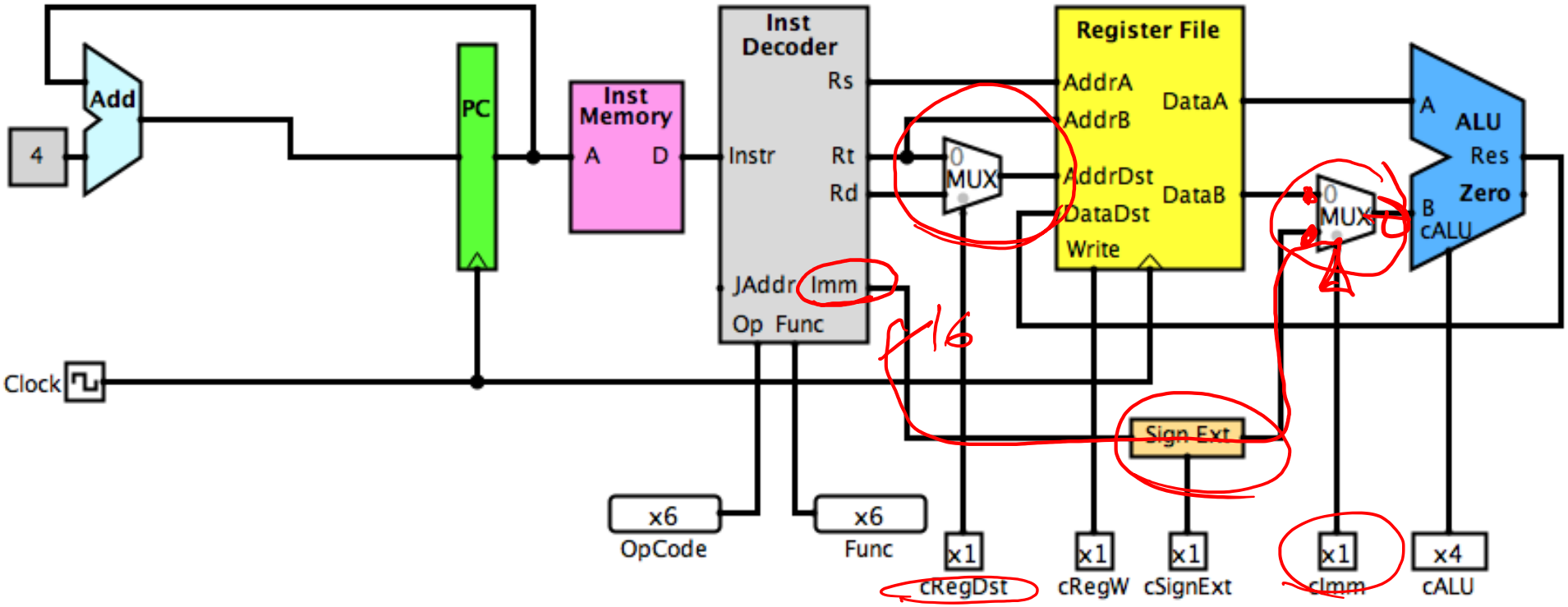
Agenda

- **MIPS Datapath**
  - add
    - instruction
    - register transfer level
    - circuit
    - timing
    - control
  - and, …
  - **addi, …**
  - beq, …
  - j
  - lw, sw
- And in Conclusion, …

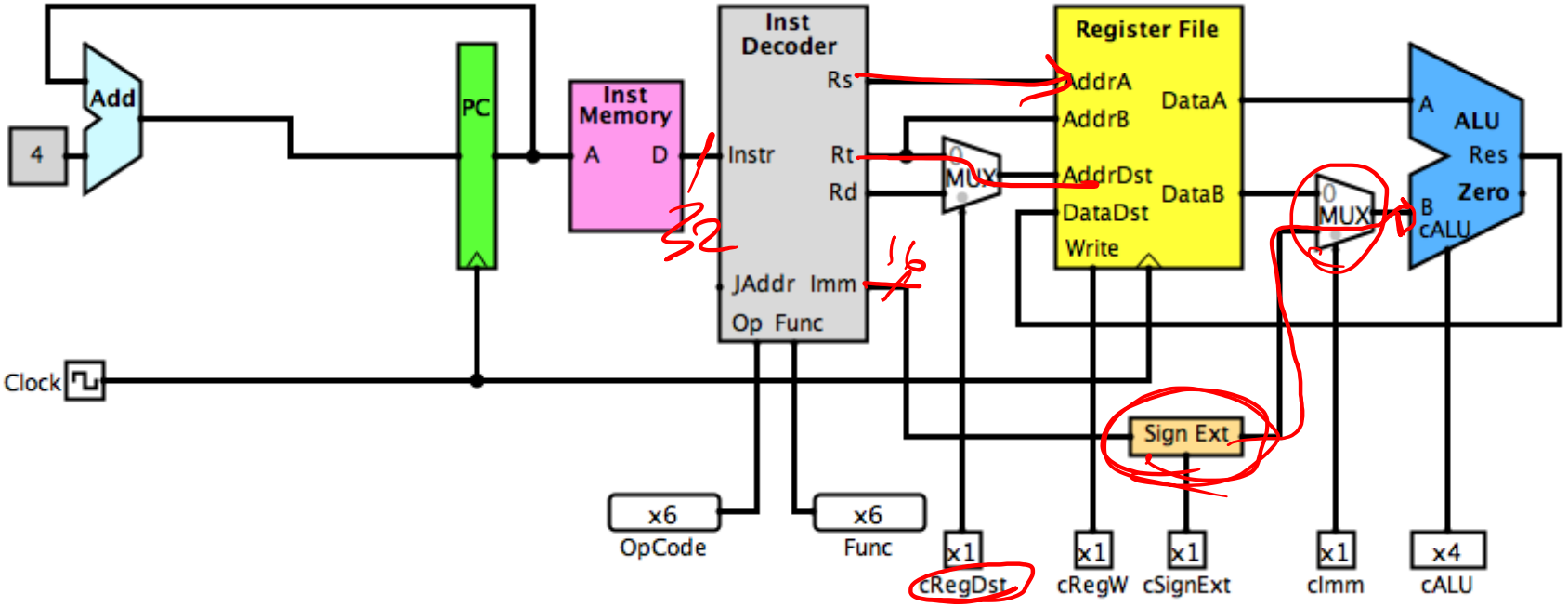
Datapath for **addiu**

Type	31 …		format (bits)	… 0
I	0x9	rs (5)	rt (5)	immediate (16)

- Register transfer level, RTL
  - `PC ← PC + 4`
  - `R[$rt] ← R[$rs] + sign_ext(imm)`
- Updated datapath to support immediate:

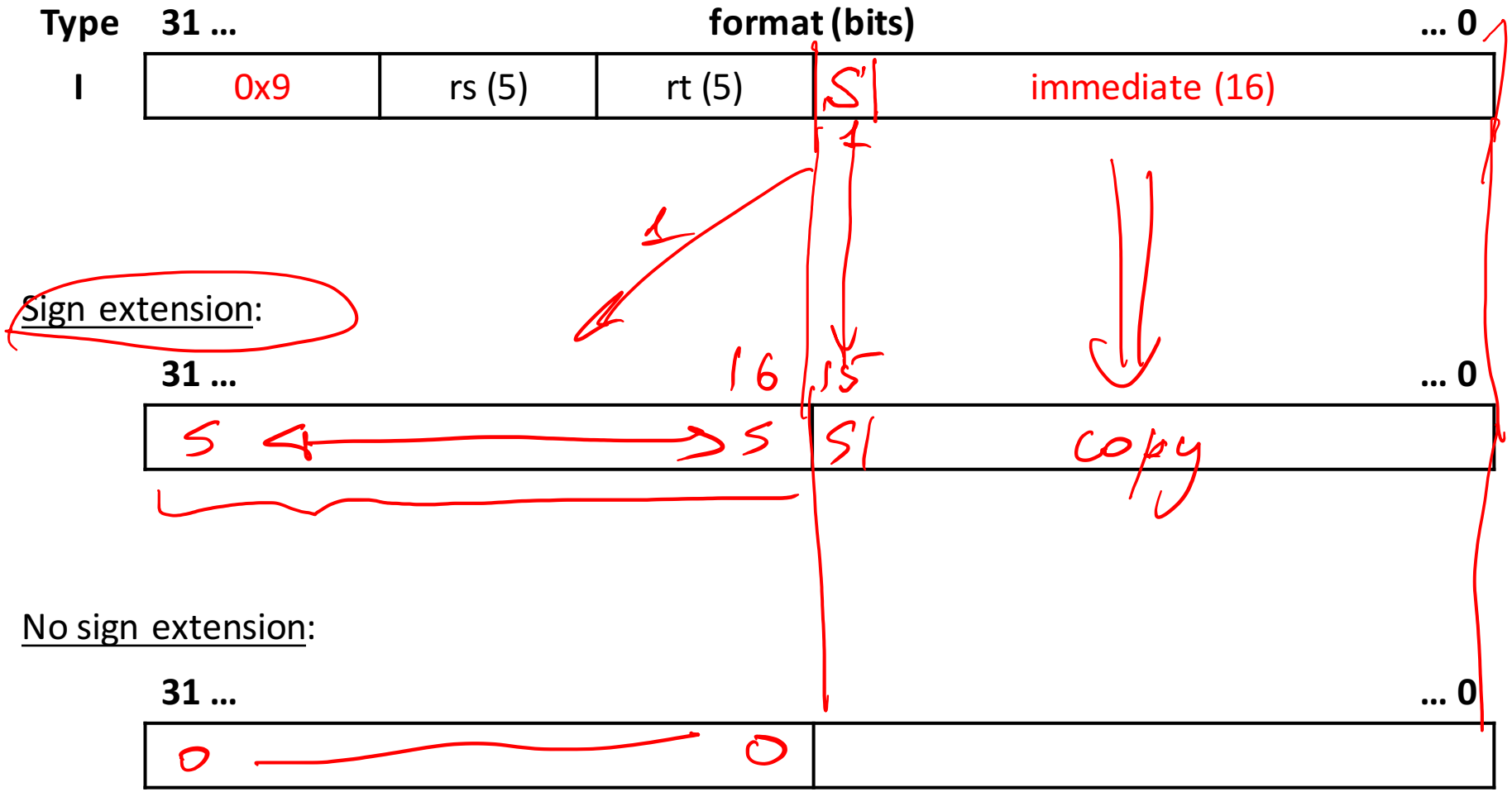


New Control Signals

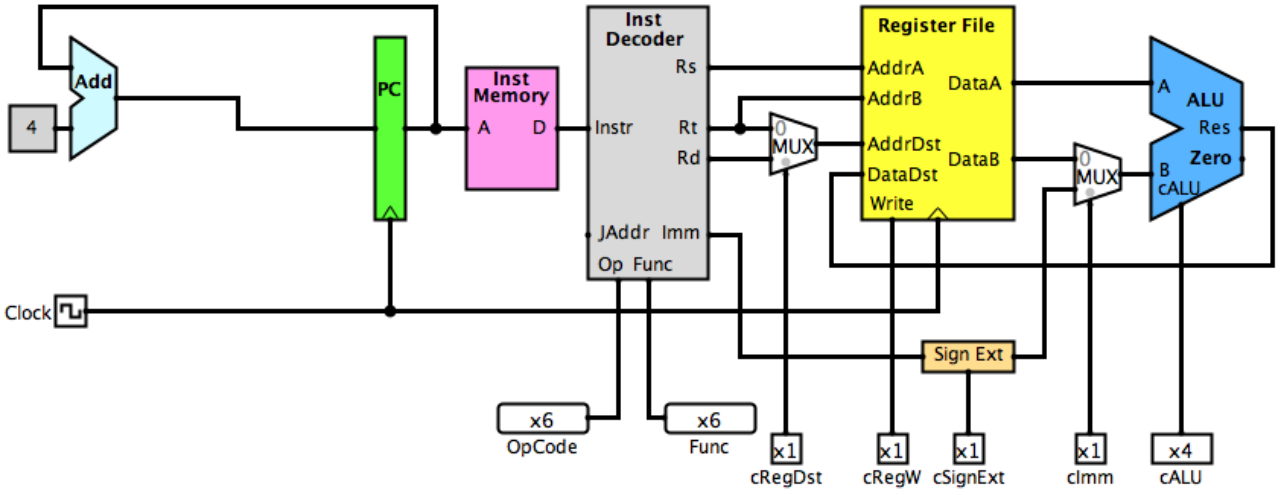


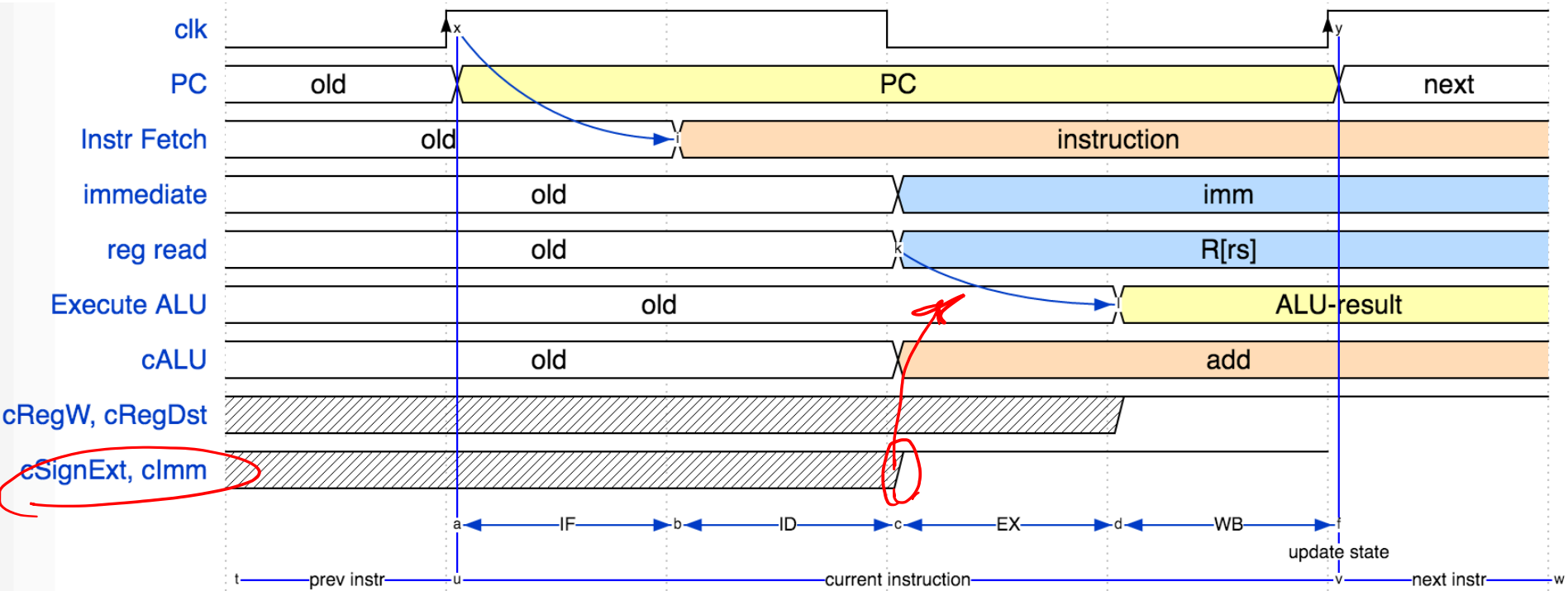
- **`cDstReg`** AddrDst = $rt (not $rd)
- **`cImm`** ALU B = immediate (not $rt)
- **`cSignExt`** Sign extend immediate

Sign Extension



Control for **addiu**

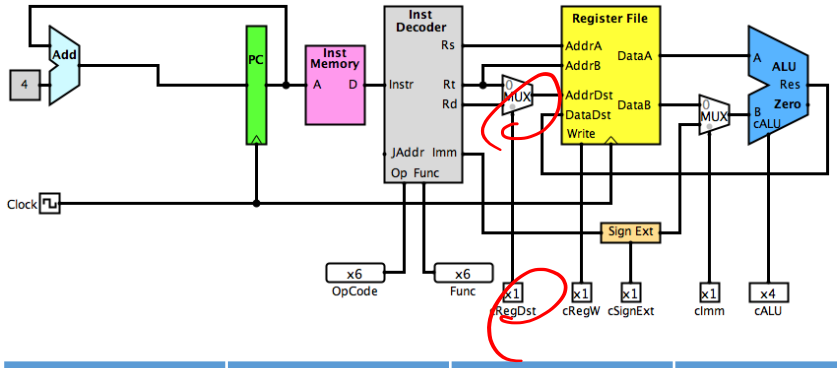


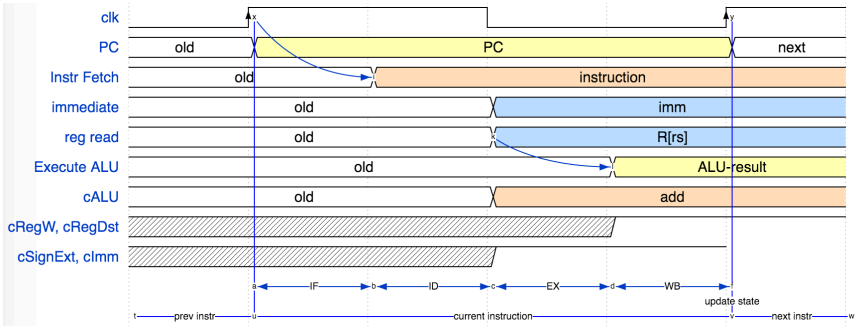


Datapath for **addiu** - Control

Type	31 …	format (bits)		… 0
I	0x9	rs (5)	rt (5)	immediate (16)

- Register transfer level, RTL
  - **PC ← PC + 4**
  - **R[$rt] ← R[$rs] + sign_ext(imm)**



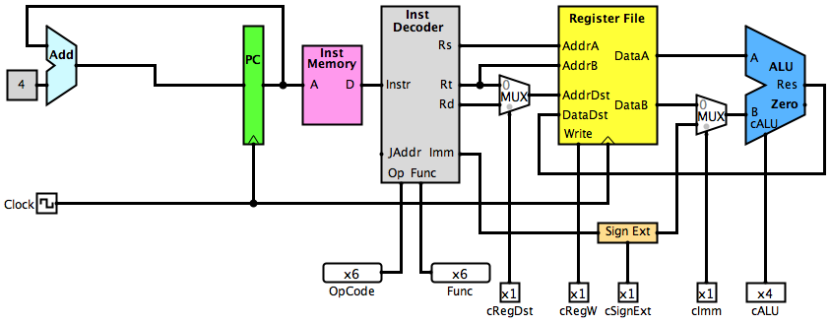


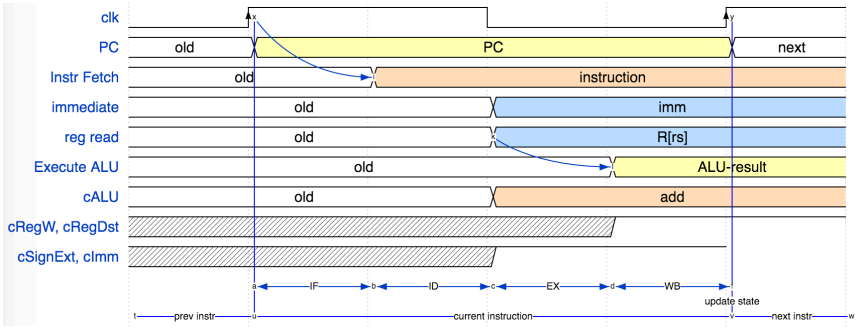
Instr	OpCode	Func	cALU	cRegW	cRegDst	cSignExt	cImm
addu	0x0	0x20	add	1	0	0	0
addiu	0x9	X	add	1	1	1	1

Datapath for **addiu** - Summary

Type	31 …	format (bits)		… 0
I	0x8	rs (5)	rt (5)	immediate (16)

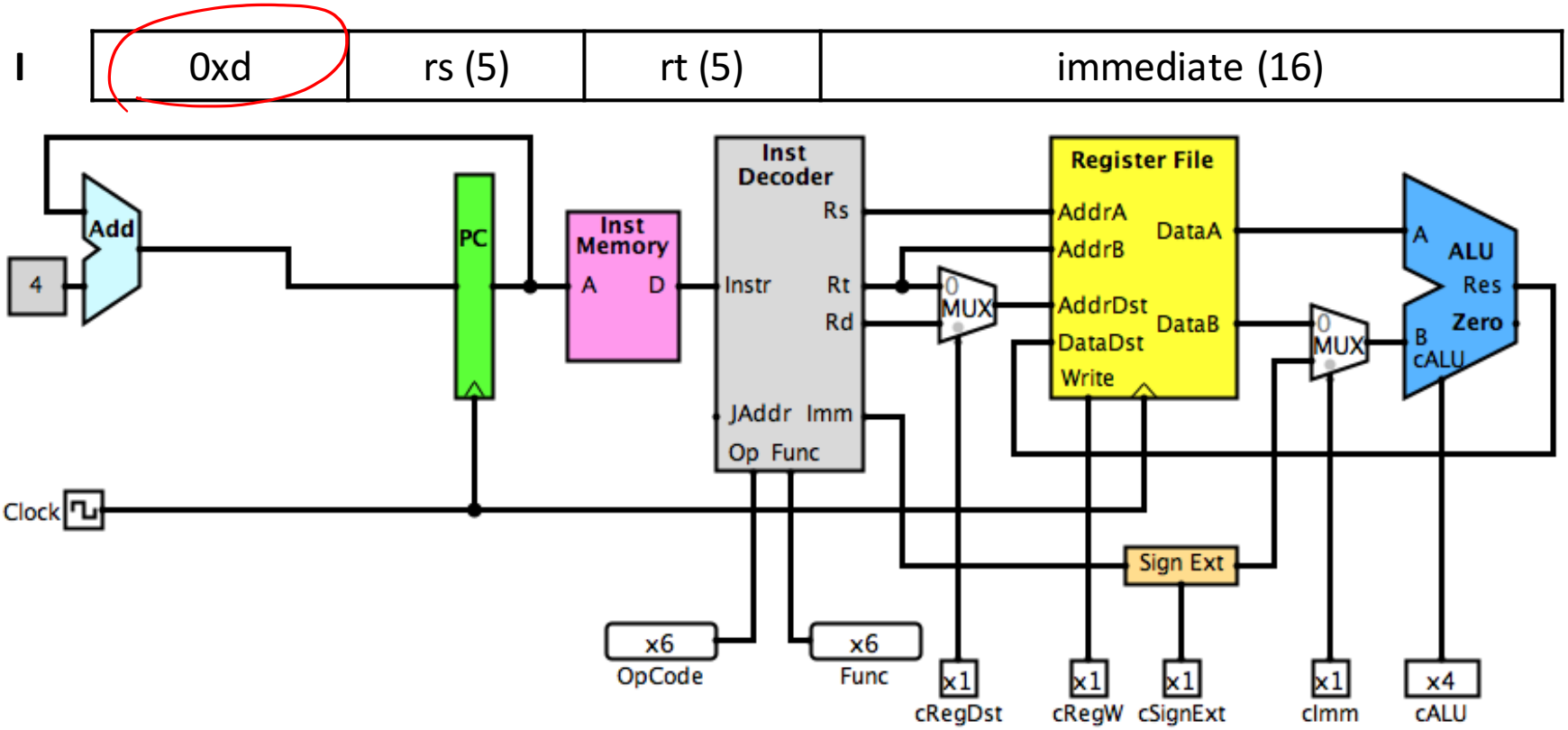
- Register transfer level, RTL
  - **PC ← PC + 4**
  - **R[$rt] ← R[$rs] + sign_ext(imm)**





Instr	OpCode	Func	cALU	cRegW	cRegDst	cSignExt	cImm
add	0x0	0x20	add	1	0	0	0
addi	0x9	X	add	1	1	1	1

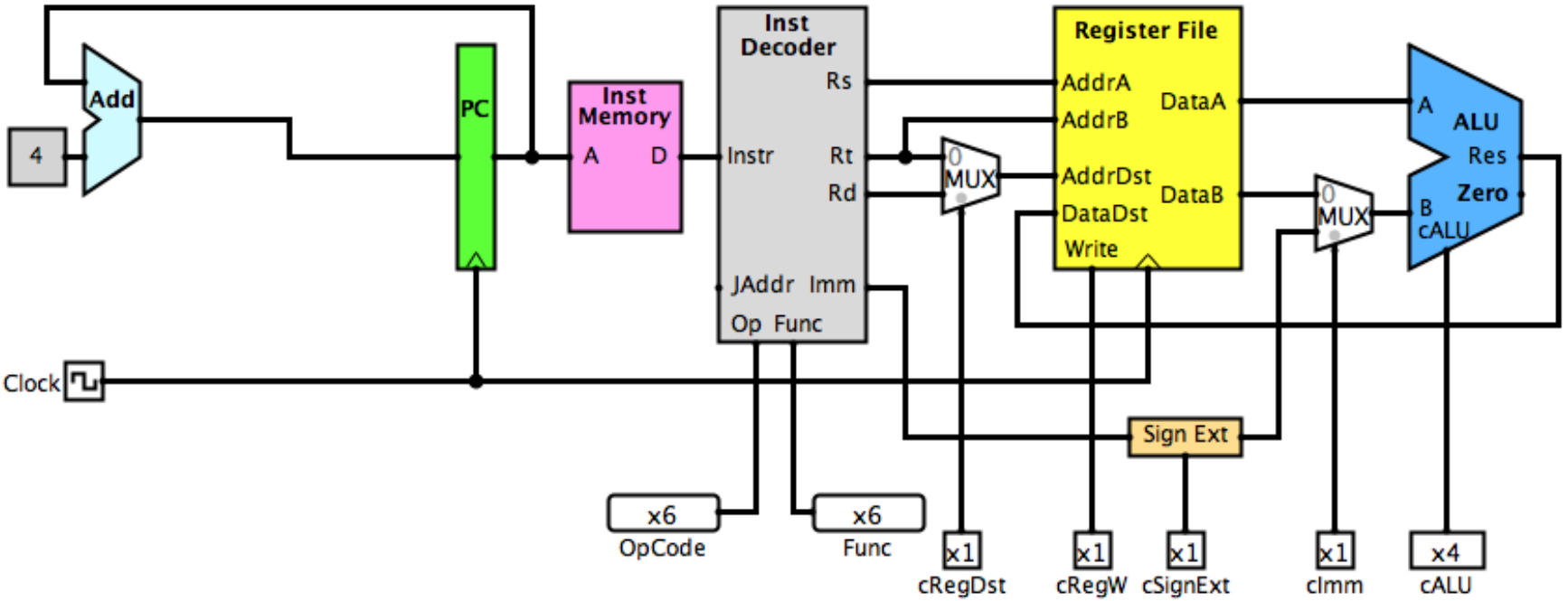
Your Turn - **ori**



Answer	OpCode	Func	cALU	cRegW	cRegDst	cSignExt	cImm
A	0xd	X	or	0	1	0	1
B	0xd	X	or	1	1	0	1
C	0x0	0xd	or	1	1	0	1
D	0xd	X	ori	1	1	1	1
E	0xd	X	or	1	1	1	1

Your Turn - **ori**

I	0xd	rs (5)	rt (5)	immediate (16)



Answer	OpCode	Func	cALU	cRegW	cRegDst	cSignExt	cImm
A	0xd	X	or	0	1	0	1
B	0xd	X	or	1	1	0	1
C	0x0	0xd	or	1	1	0	1
D	0xd	X	ori	1	1	1	1
E	0xd	X	or	1	1	1	1

Gorilla Tutoring

- Gorilla Sessions
  - Wednesdays 7:30 – 9:30 pm
  - 405 Soda or 293 Cory
  - 10 tutors
  - review & practice questions
- Private Tutoring
  - 2 tutors
  - 10 students max
  - Sign-up instructions on Piazza

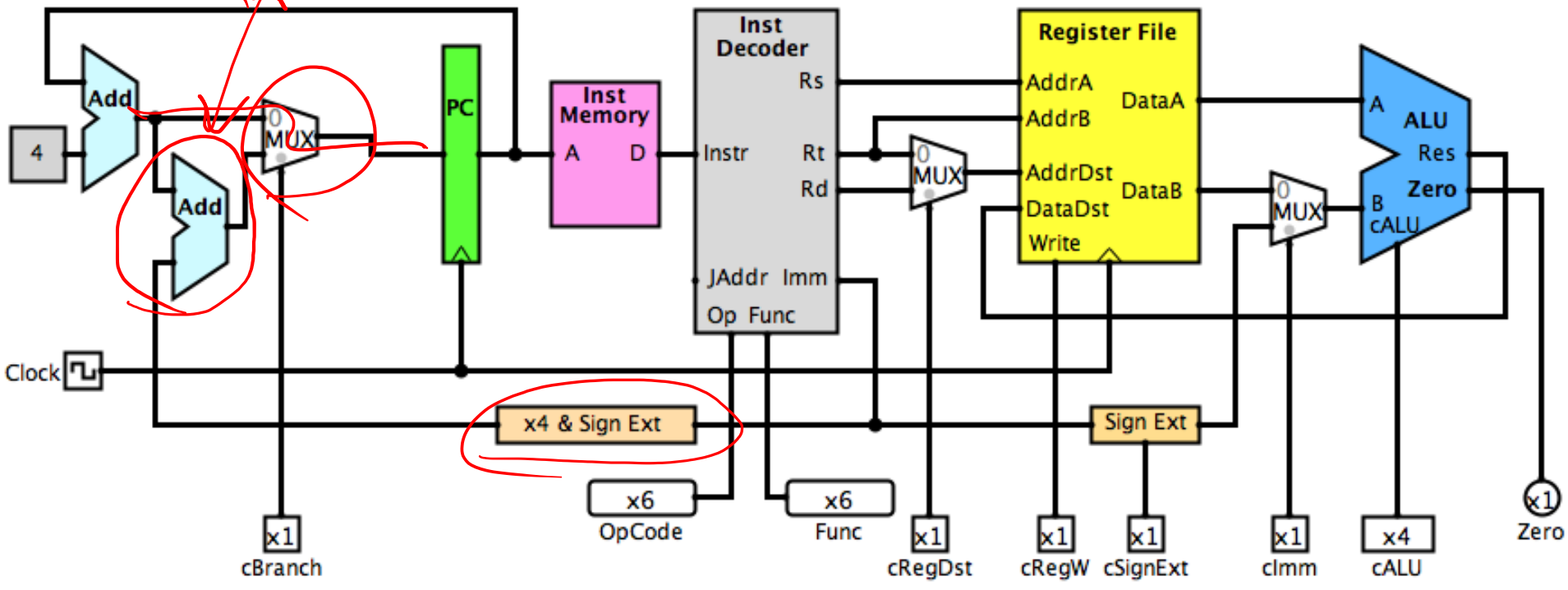
Agenda

- **MIPS Datapath**
  - add
    - instruction
    - register transfer level
    - circuit
    - timing
    - control
  - and, …
  - addi, ...
  - **beq, …**
  - j
  - lw, sw
- And in Conclusion, ...

Datapath for **beq**

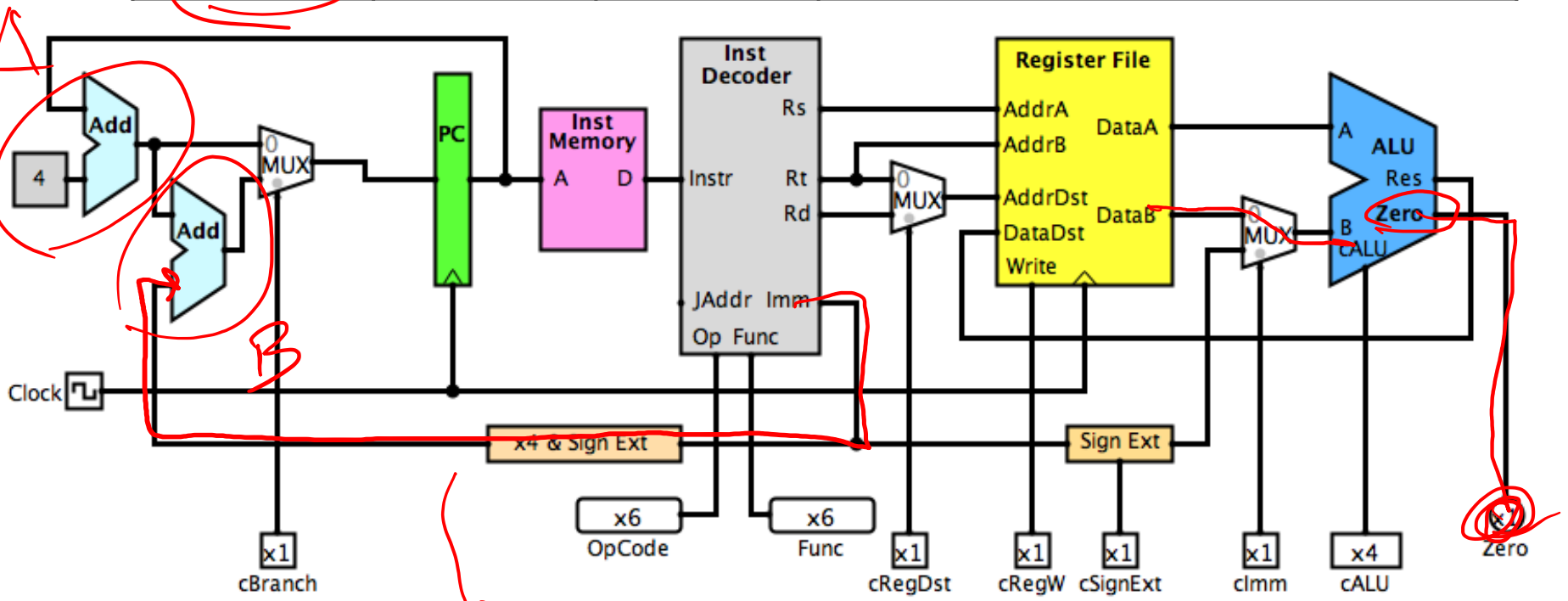
Type	31 …	format (bits)		… 0
I	0x5	rs (5)	rt (5)	offset (16)

- Register transfer level, RTL
  - `PC ← PC + 4`
    `+ if (R[$rs]==R[$rt]) then signext(4*off)`
    `else 0`



Control for **beq**

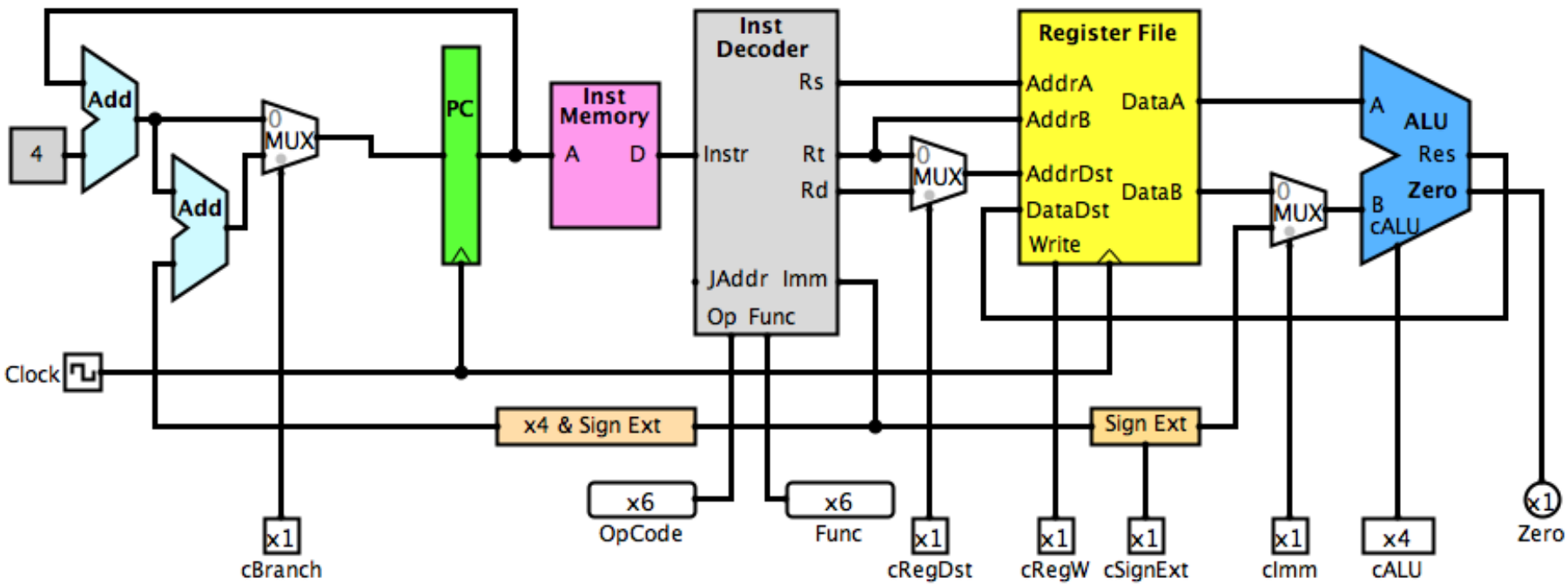
Type	31 …	format (bits)		… 0
I	0x4	rs (5)	rt (5)	offset (16)



Instr	OpCode	Func	Zero	cALU	cRegW	cRegDst	cSignExt	cImm	cBranch
addiu	0x9	X	X	add	1	1	1	1	0
addu	0x0	0x21	X	add	1	0	0	0	0
beq	0x4	X	1	sub	0	X	X	0	1
	0x4	X	0	sub	0	X	X	0	0

Control for **beq**

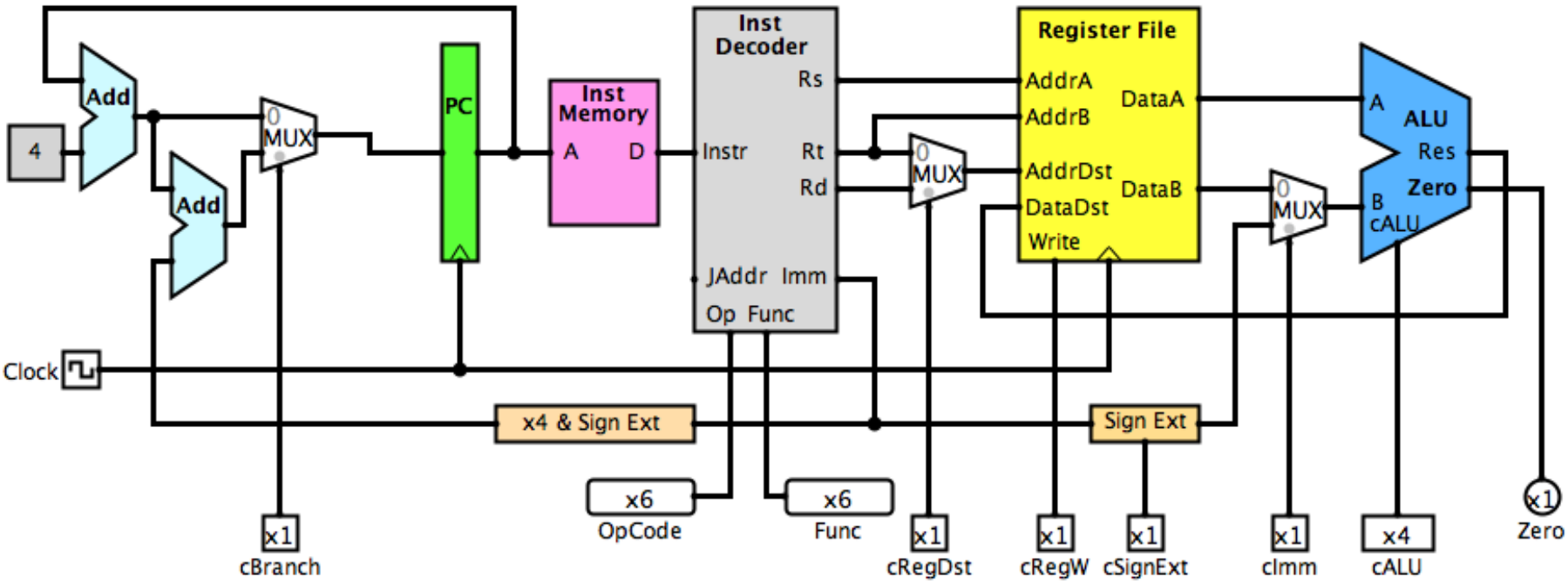
Type	31 …	format (bits)	… 0		
I	0x5	rs (5)	rt (5)	offset (16)	



Instr	OpCode	Func	Zero	cALU	cRegW	cRegDst	cSignExt	cImm	cBranch
addi	0x8	X	X	add	1	1	1	1	0
add	0x0	0x20	X	add	1	0	0	0	0
beq	0x4	X	1	sub	0	X	X	0	1
beq	0x4	X	0	sub	0	X	X	0	0

Datapath for **bneq**

Type	31 …	format (bits)		… 0
I	0x5	rs (5)	rt (5)	offset (16)



Instr	OpCode	Func	Zero	cALU	cRegW	cRegDst	cSignExt	cImm	cBranch
beq	0x4	X	1	sub	0	X	X	0	1
beq	0x4	X	0	sub	0	X	X	0	0
bneq	0x5								

Agenda

- **MIPS Datapath**
  - add
    - instruction
    - register transfer level
    - circuit
    - timing
    - control
  - and, …
  - addi, ...
  - beq, …
  - **j**
  - lw, sw
- And in Conclusion, ...

Datapath for `j`

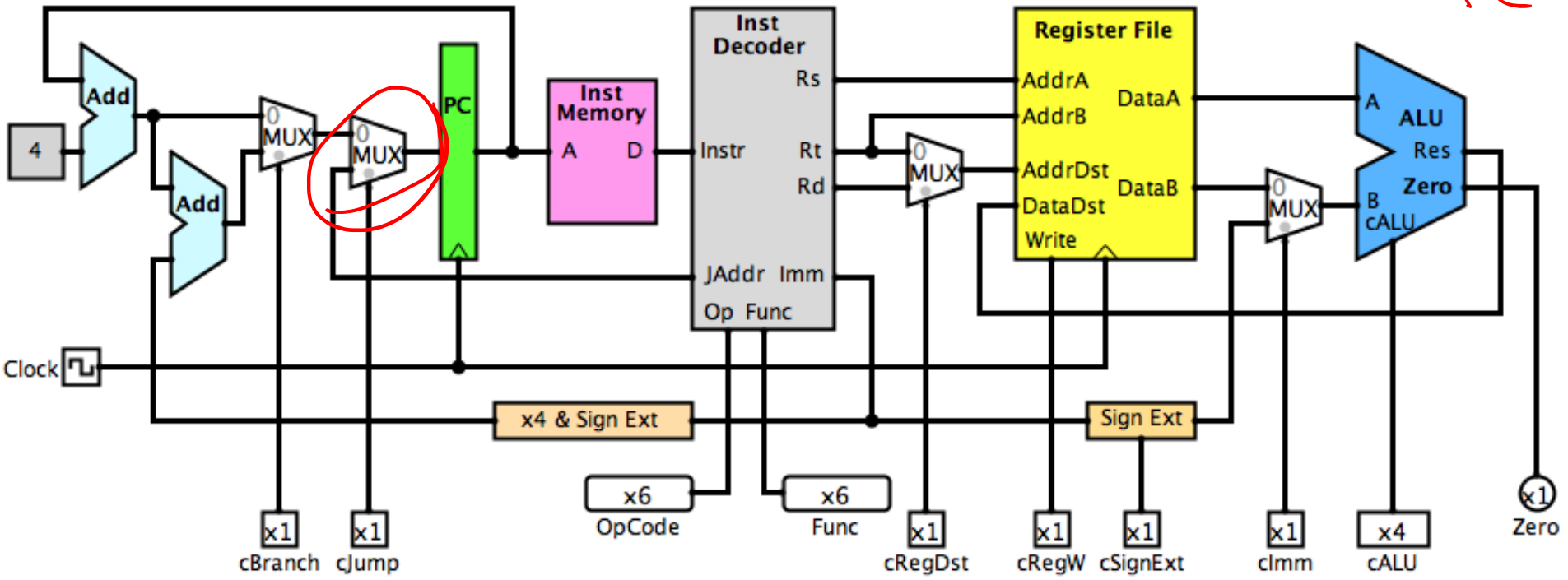
Type	31 …	format (bits)	… 0
J	0x2	target address (26)	

- Register transfer level, RTL

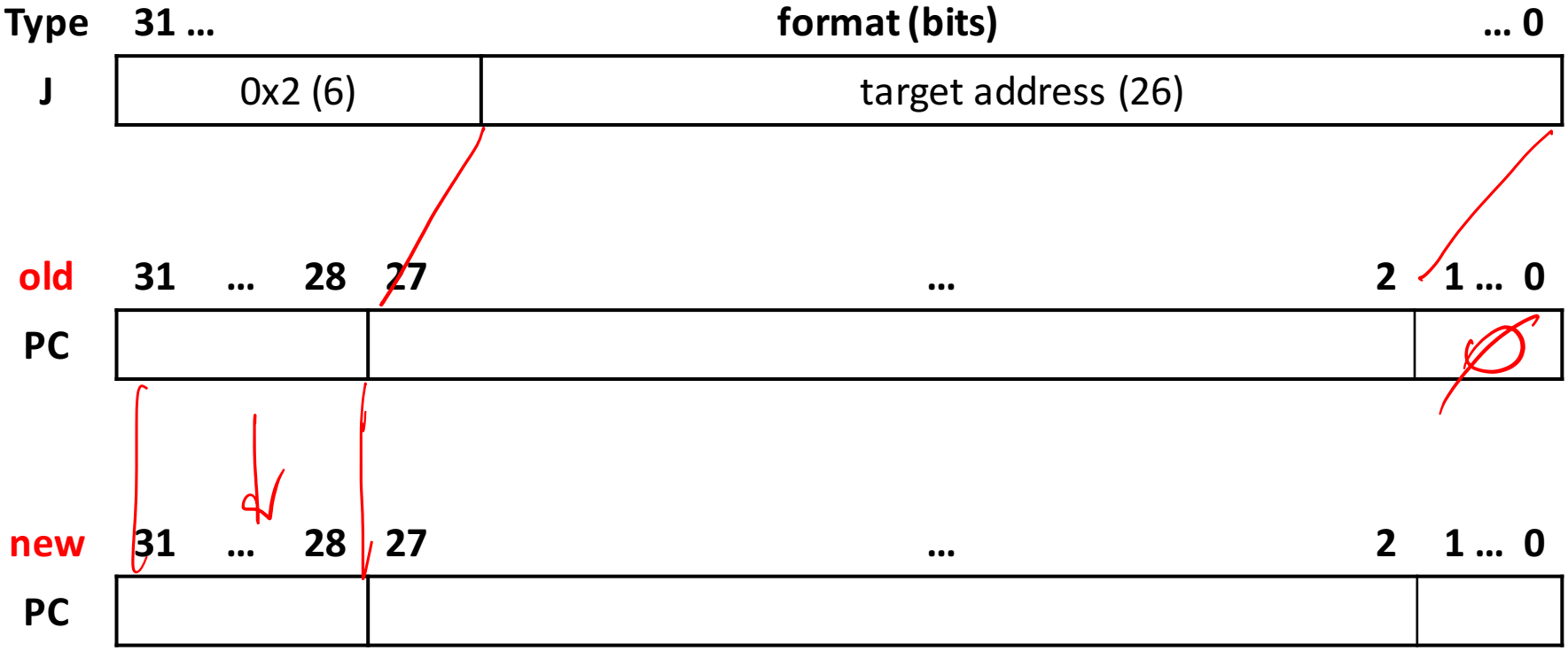
`PC[2 … 28] ← target address`

PC[0..1] ← 0

PC[28...31] ← OLD PC

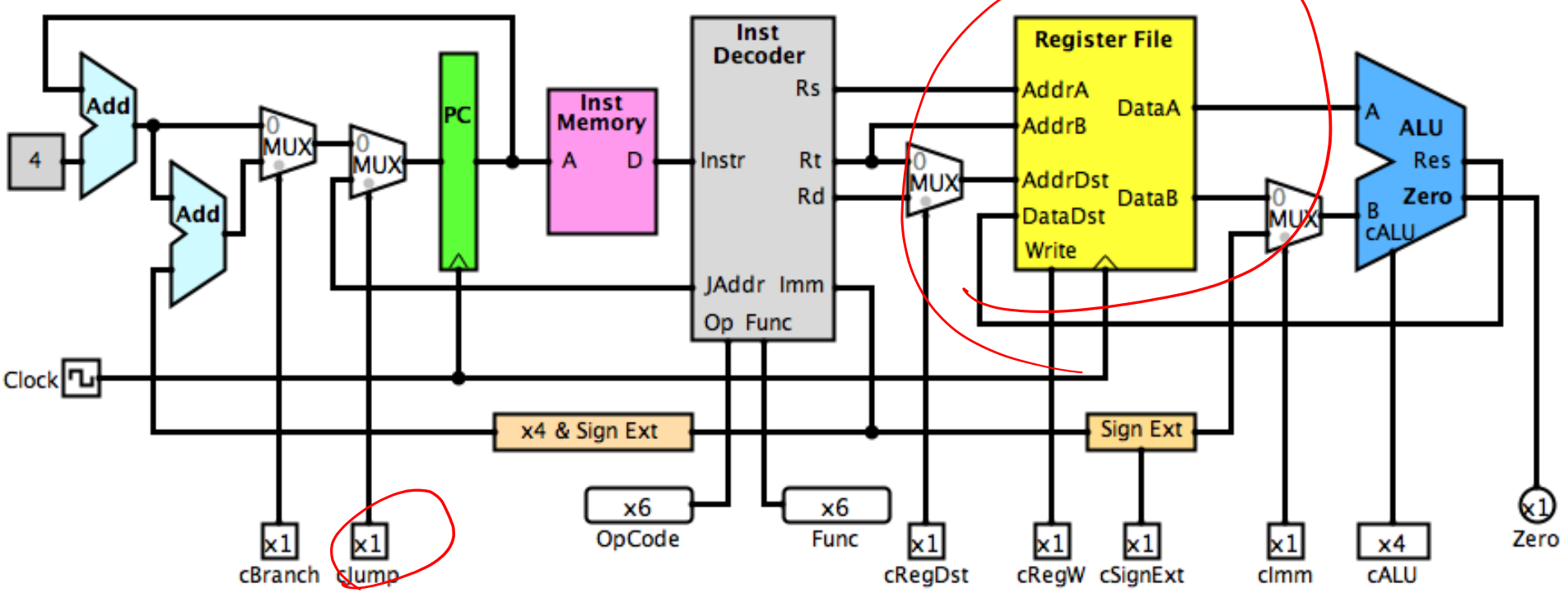


j Destination address



Control for `j`

Type	31 …	format (bits)	… 0
J	0x2	target address (26)	



Instr	OpCode	Func	Zero	cALU	cRegW	cRegDst	cSignExt	cImm	cBranch	cJump
addiu	0x9	X	X	add	1	1	1	1	0	0
addu	0x0	0x21	X	add	1	0	0	0	0	0
j	**0x2**	**X**	**X**	**X**	**0**	**X**	**X**	**X**	**X**	**1**

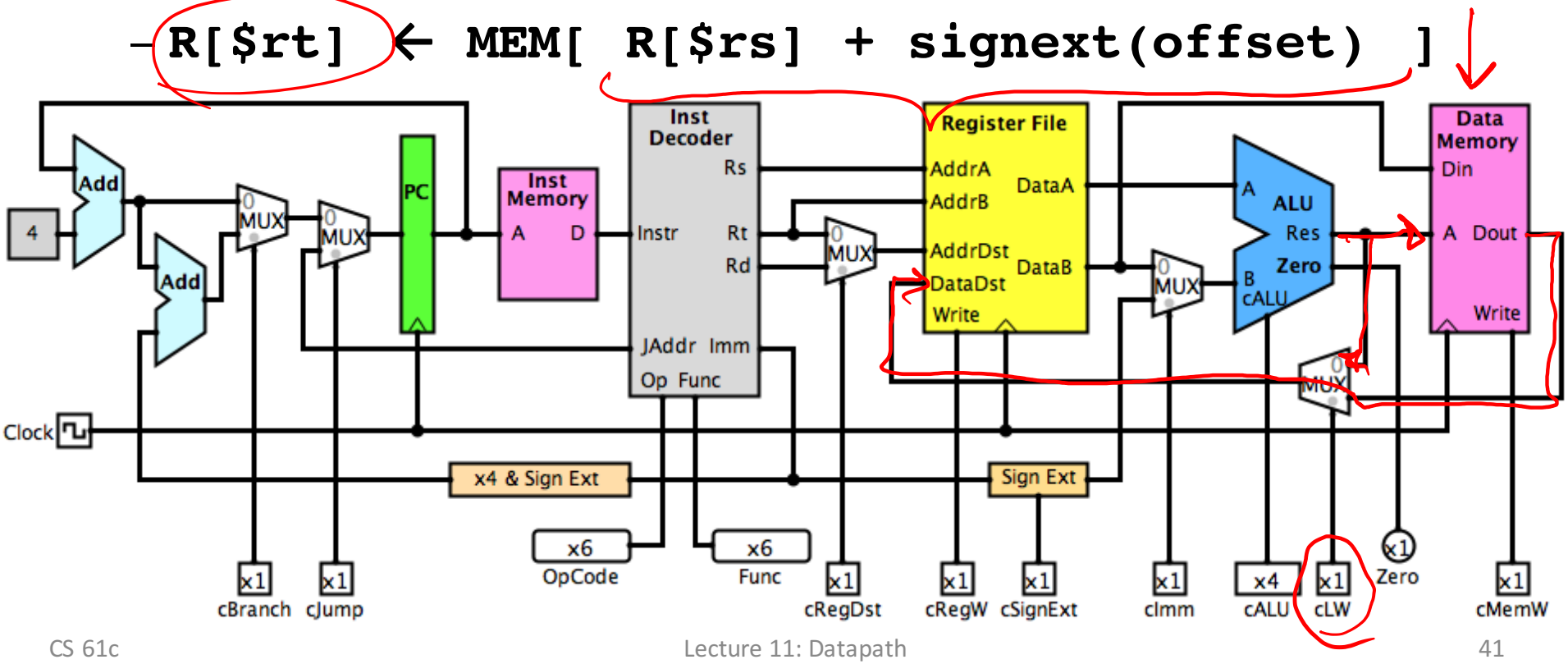
Agenda

- **MIPS Datapath**
  - add
    - instruction
    - register transfer level
    - circuit
    - timing
    - control
  - and, …
  - addi, ...
  - beq, …
  - j
  - **lw, sw**
- And in Conclusion, ...

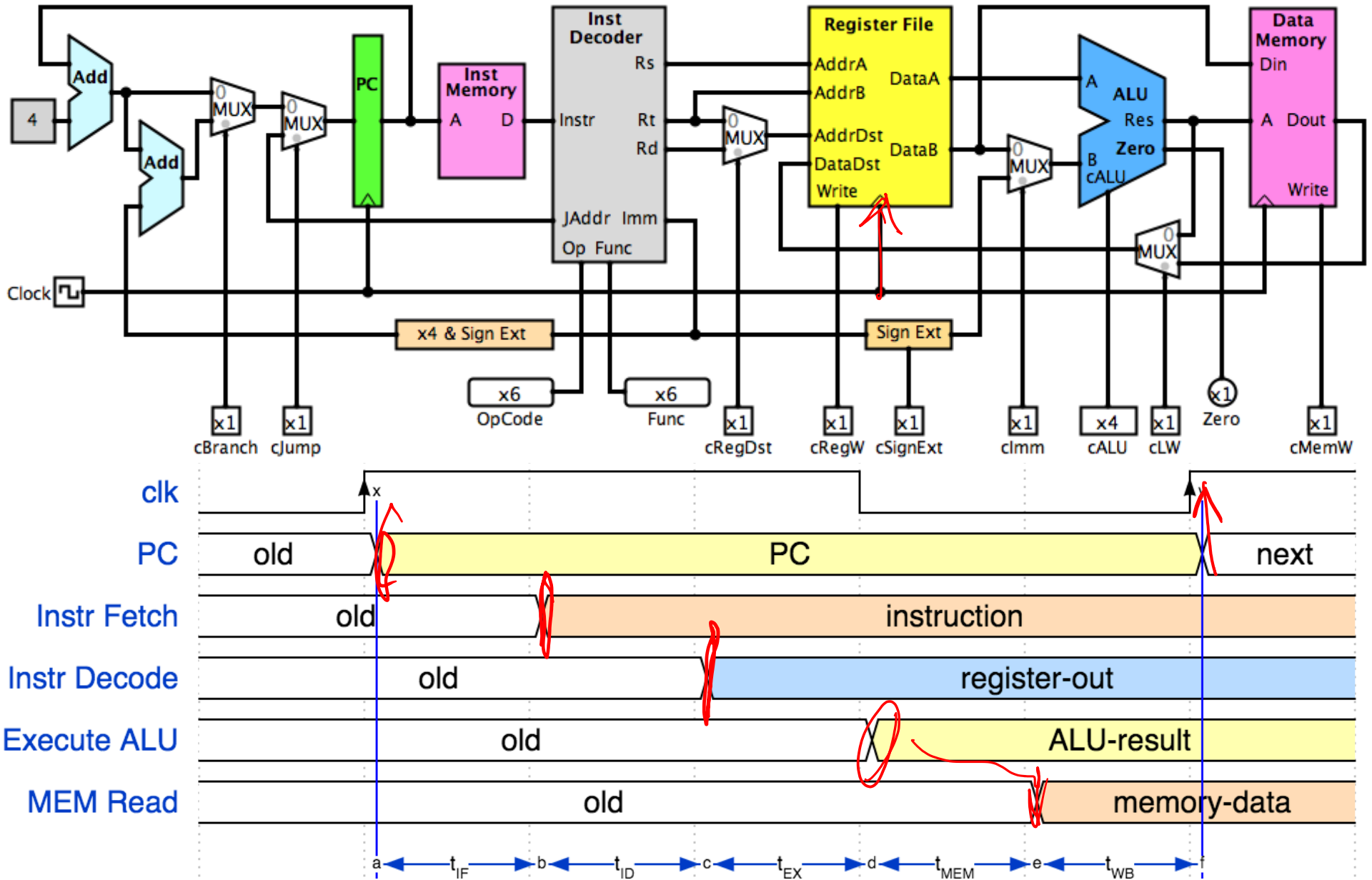
Datapath for **lw**

Type	31 …	format (bits)		… 0
I	0x23	rs (5)	rt (5)	offset (16)

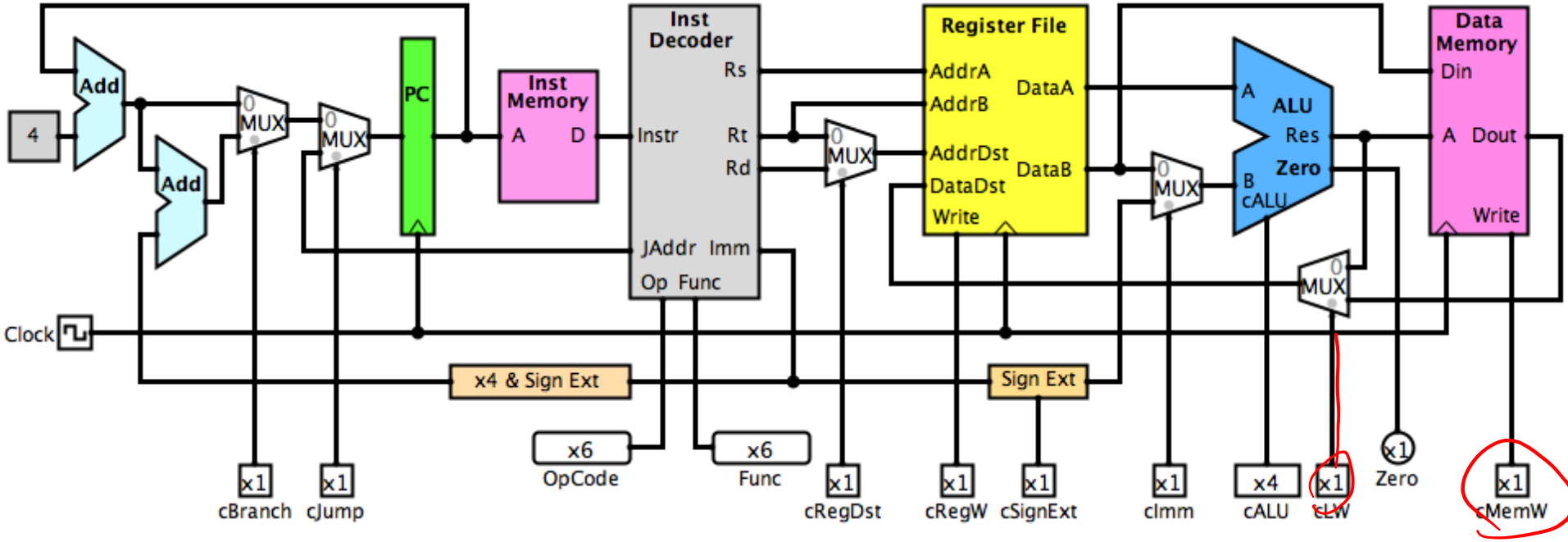
- Register transfer level, RTL
  - **PC ← PC + 4**
  - **R[$rt] ← MEM[R[$rs] + signext(offset)]**



Timing for **`lw`**

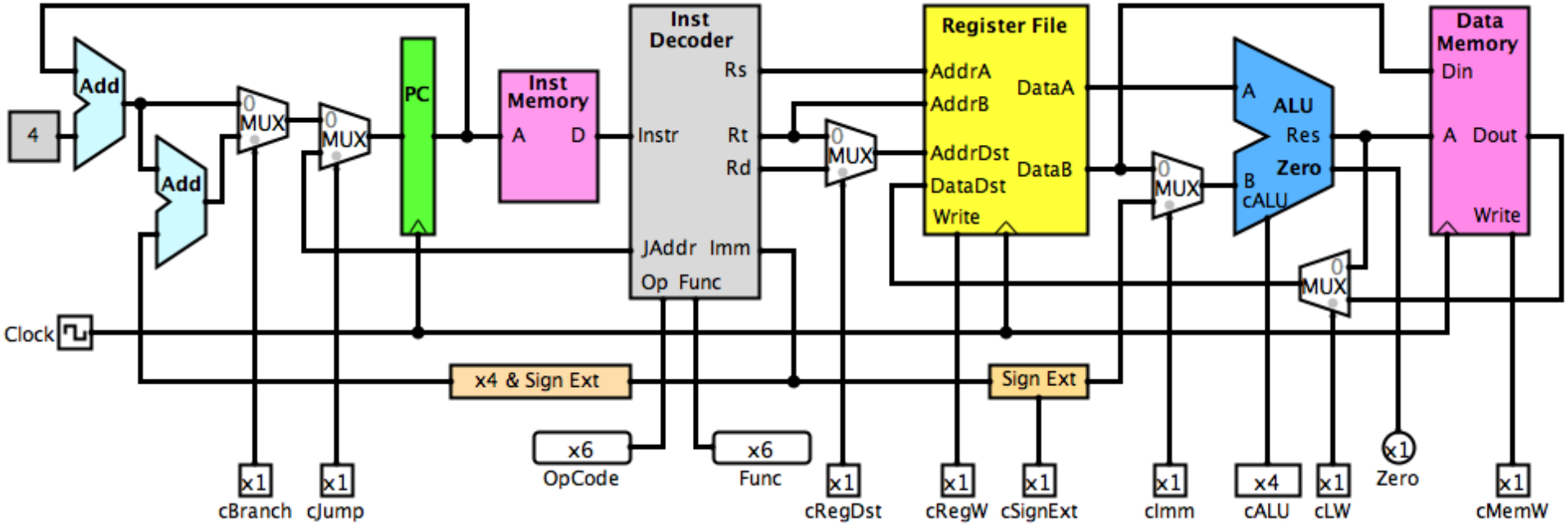


Control for **lw**



Instr	OpCode	Func	Zero	cALU	cRegW	cRegDst	cSignExt	cImm	cBr	cJ	cLW	cMemW
addiu	0x9	X	X	add	1	1	1	1	0	0	0	0
addu	0x0	0x21	X	add	1	0	0	0	0	0	0	0
j	0x2	X	X	X	0	X	X	X	X	1	X	0
lw	**0x23**	X	X	add	1	1	1	1	0	0	1	0

Control for **lw**

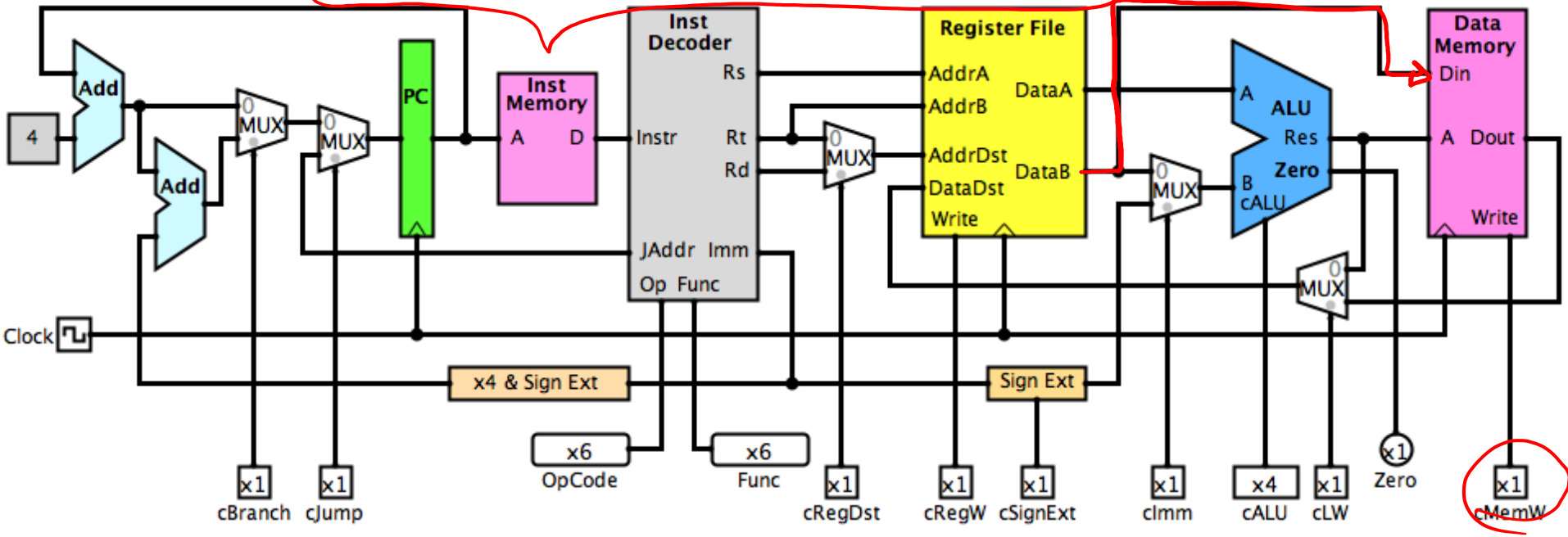


Instr	OpCode	Func	Zero	cALU	cRegW	cRegDst	cSignExt	cImm	cBr	cJ	cLW	cMemW
addiu	0x9	X	X	add	1	1	1	1	0	0	0	0
addu	0x0	0x21	X	add	1	0	0	0	0	0	0	0
j	0x2	X	X	X	0	X	X	X	X	1	X	0
lw	**0x23**	**X**	**X**	**add**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**0**

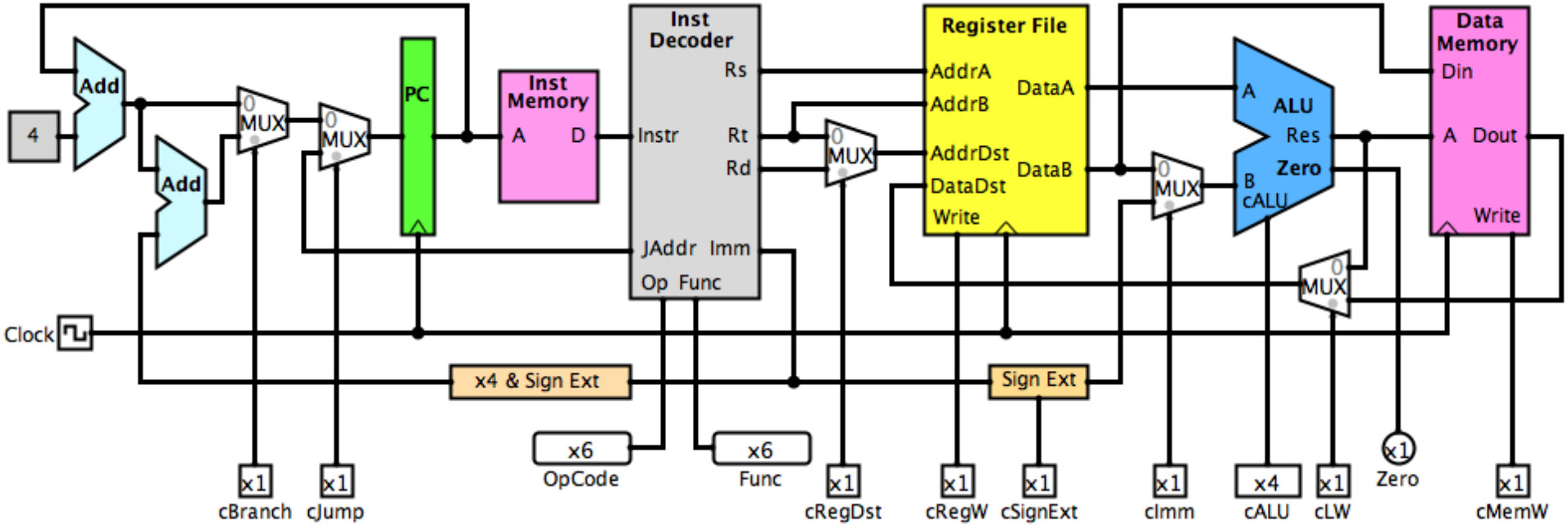
Datapath for **sw**

Type	31 …	format (bits)		… 0
I	0x15	rs (5)	rt (5)	offset (16)

- Register transfer level, RTL
  - **PC ← PC + 4**
  - **MEM[R[$rs] + signext(offset)] ← R[$rt]**

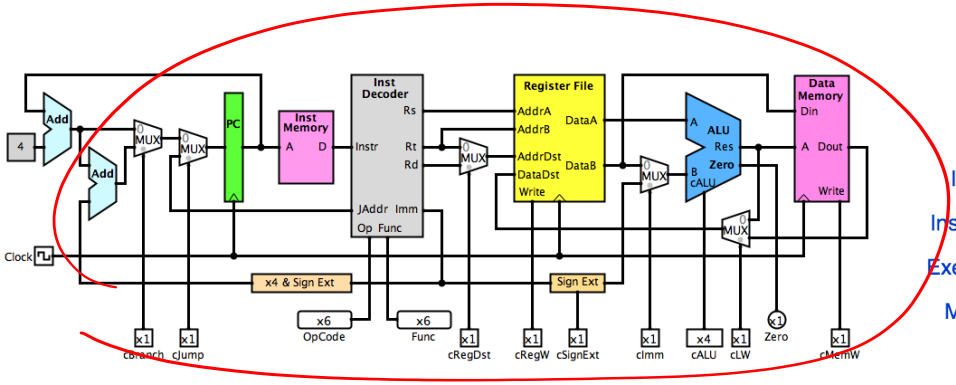


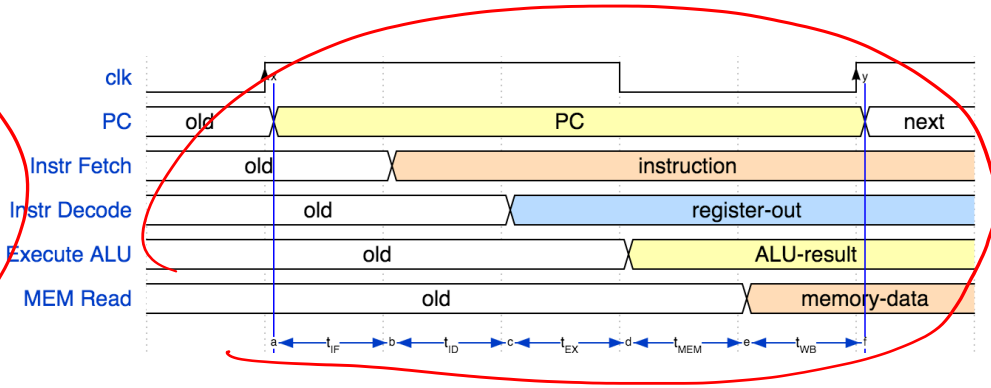
Control for **sw**



Instr	OpCode	Func	Zero	cALU	cRegW	cRegDst	cSignExt	cImm	cBr	cJ	cLW	cMemW
addiu	0x9	X	X	add	1	1	1	1	0	0	0	0
addu	0x0	0x21	X	add	1	0	0	0	0	0	0	0
j	0x2	X	X	X	0	X	X	X	X	1	X	0
lw	0x23	X	X	add	**1**	1	1	1	0	0	**1**	**0**
sw	0x2b	X	X	add	**0**	X	1	1	0	0	**X**	**1**

MIPS Datapath Summary





Phase	Action	Time Slot	Hardware Used
1	Fetch instruction	t_{IF}	Instr Memory
2	Decode instr, read registers	t_{ID}	Instr Dec, Register File
3	Execute instruction	t_{EX}	ALU
4	Access data memory	t_{MEM}	Data Memory
5	Write register, PC	t_{WB}	Register File, PC

Note: not all instructions are active in all phases

Your Turn

Which instruction is active in the fewest/most phases?

Answer	Fewest	Most
A	`nor`	`lw`
B	`bne`	`sw`
C	`j`	`lw`
D	`jr`	`lw`
E	`slt`	`sw`

Your Turn

Which instruction is active in the fewest/most phases?

Answer	Fewest	Most
A	`nor`	`lw`
B	`bne`	`sw`
C	`j`	`lw`
D	`jr`	`lw`
E	`slt`	`sw`

Agenda

- **MIPS Datapath**
  - add
    - instruction
    - register transfer level
    - circuit
    - timing
    - control
  - and, …
  - addi, ...
  - beq, …
  - j
  - lw, sw
  - **dowhile**
- And in Conclusion, ...

MIPS 2000, 4000, MIPS 61C!

```
int j = -1;
do
        // a[i] = b[i] * c[i]; …
    i = i+j;
while (i != 0);
```

New instruction that updates i and branches:

`dowhile $i, $j, label`

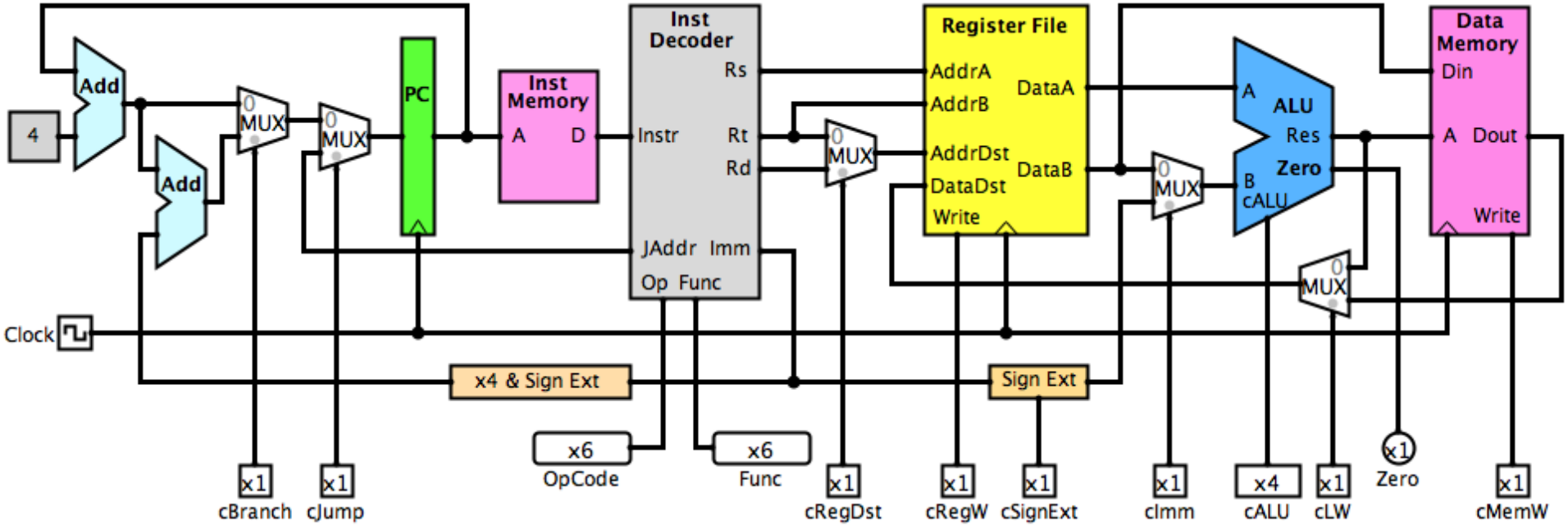
dowhile $rt, $rs, label

Type	31 …		format (bits)	… 0
I	*0x61c**	rs (5)	rt (5)	offset (16)

*just replace one of the less useful instructions …

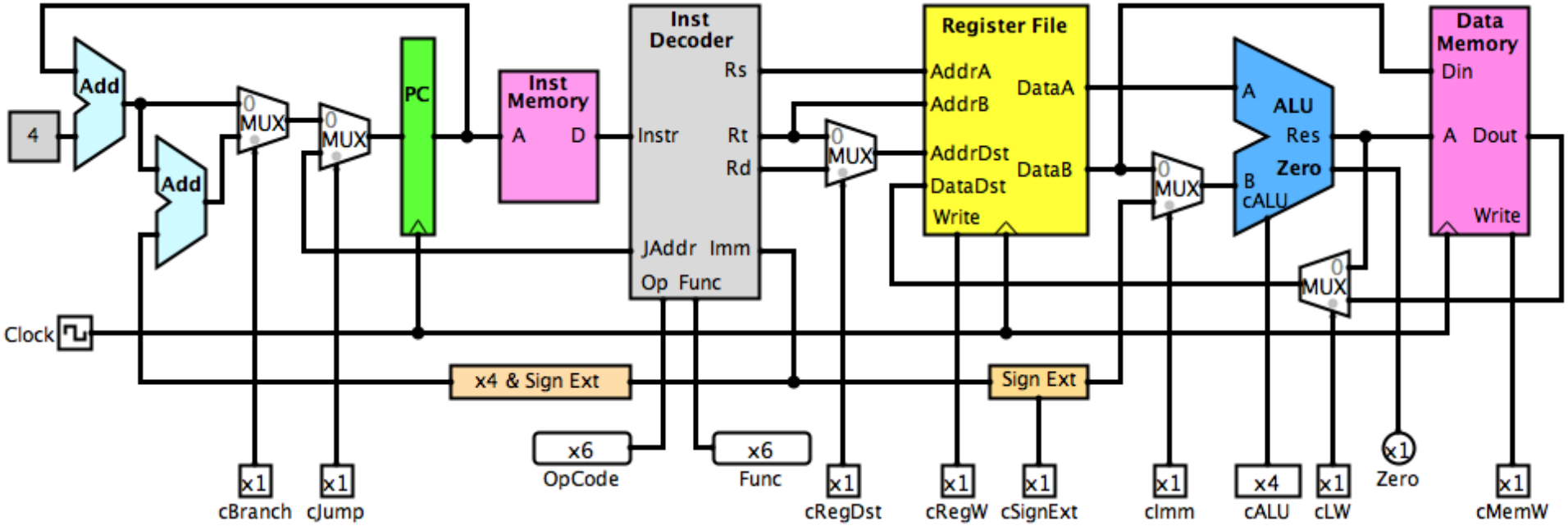
- Register transfer level, RTL
  - **R[$rt] = R[$rt] + R[$rs]**
  - **PC ← PC + 4**
    + if (R[$rs]==R[$rt]) then signext(4*off)
    else 0

Control for **dowhile**



Instr	OpCode	Zero	cALU	cRegW	cRegDst	cSignExt	cImm	cBr	cJ	cLW	cMemW
dowhile	*0x61c*										

Control for **dowhile**



Instr	OpCode	Zero	cALU	cRegW	cRegDst	cSignExt	cImm	cBr	cJ	cLW	cMemW
dowhile	*0x61c*	1	add	1	1	X	0	1	0	0	0
		0	add	1	1	X	0	0	0	0	0

Agenda

- MIPS Datapath
  - add
    - instruction
    - register transfer level
    - circuit
    - timing
    - control
  - and, …
  - addi, ...
  - beq, …
  - j
  - lw, sw
- **And in Conclusion, …**

And in Conclusion, …

- Universal datapath
  - Capable of executing all MIPS instructions
  - Not all units (hardware) used by all instructions
- 5 Phases of execution
  - IF, ID, EX, MEM, WB
  - Not all instructions are active in all phases
- Controller specifies how to execute instructions
  - Even "new" ones … (**`dowhile`**)