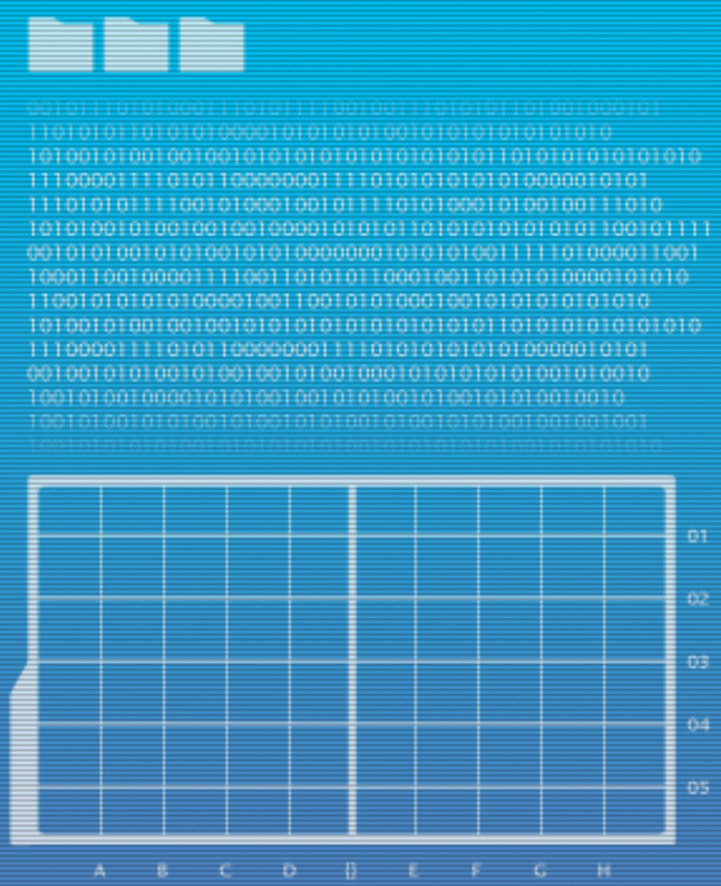
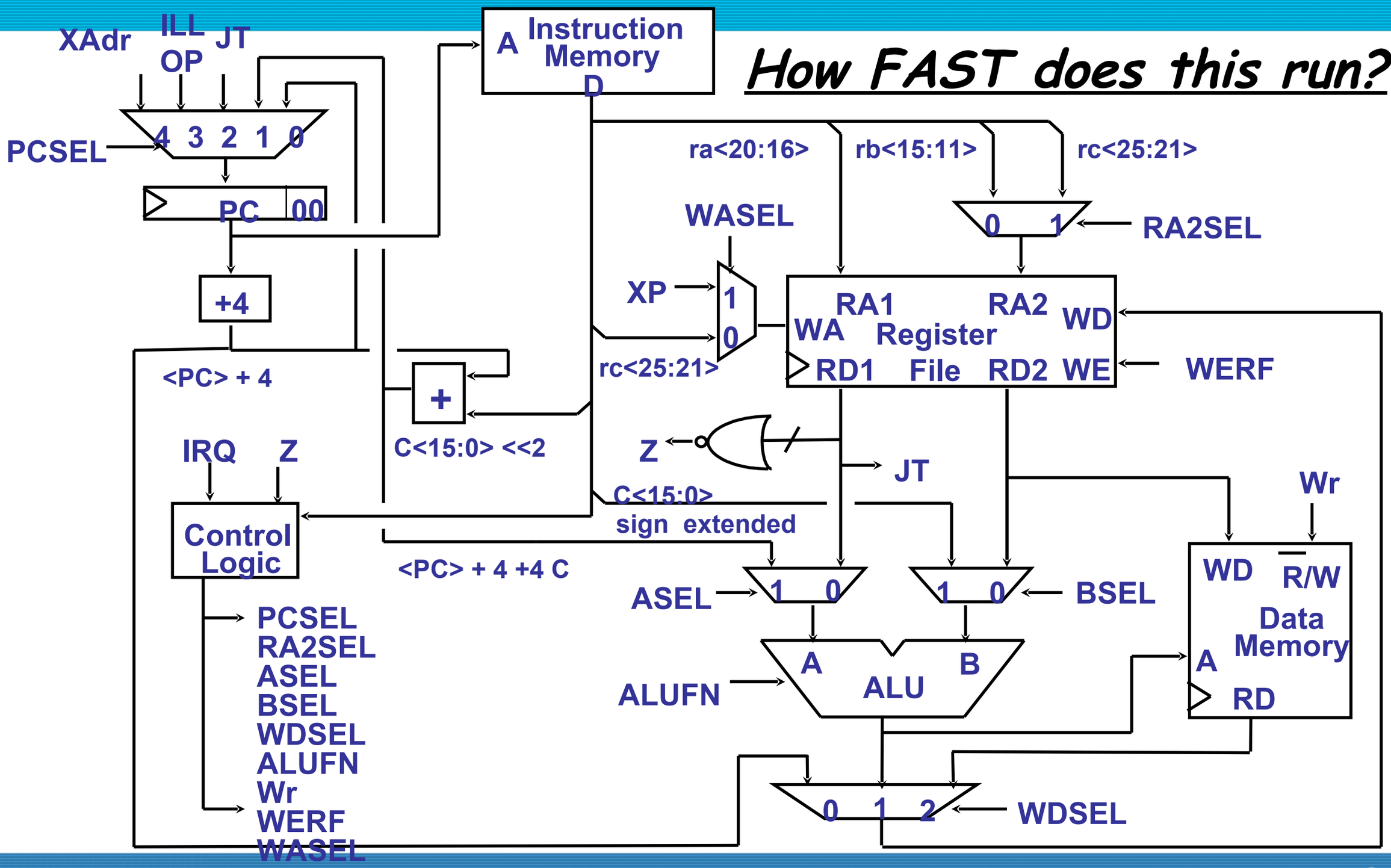
DEPARTMENT OF
INFORMATION SYSTEMS
AND COMPUTER SCIENCE



Pipelining the Beta

Speed at what Cost?

CPU Performance



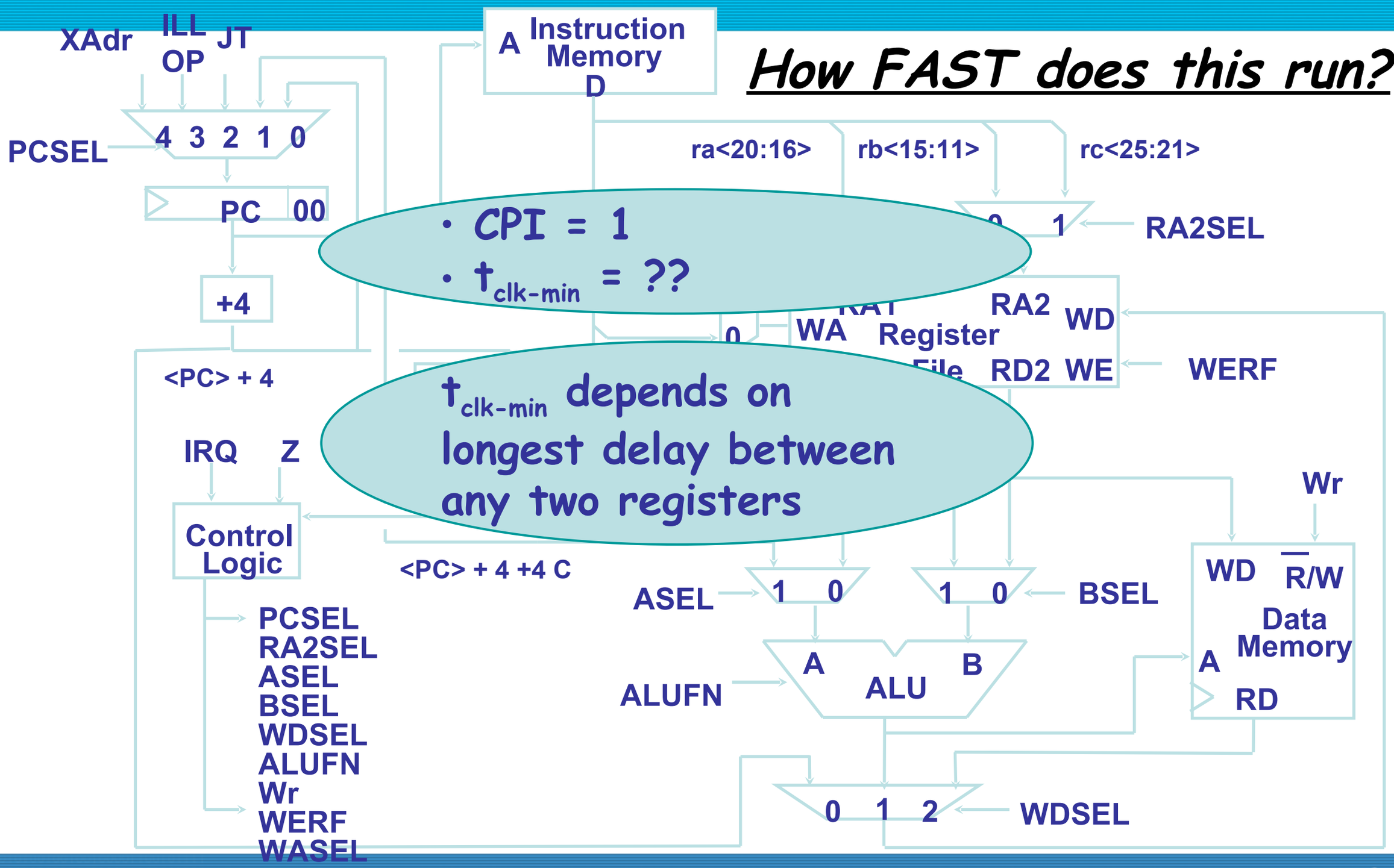
CPU Performance

- MIPS: Millions of Instructions Per Second
  - Common measure of performance:

$$\text{MIPS} = \frac{\text{Frequency in MHz}}{\text{Clocks Per Instruction}}$$

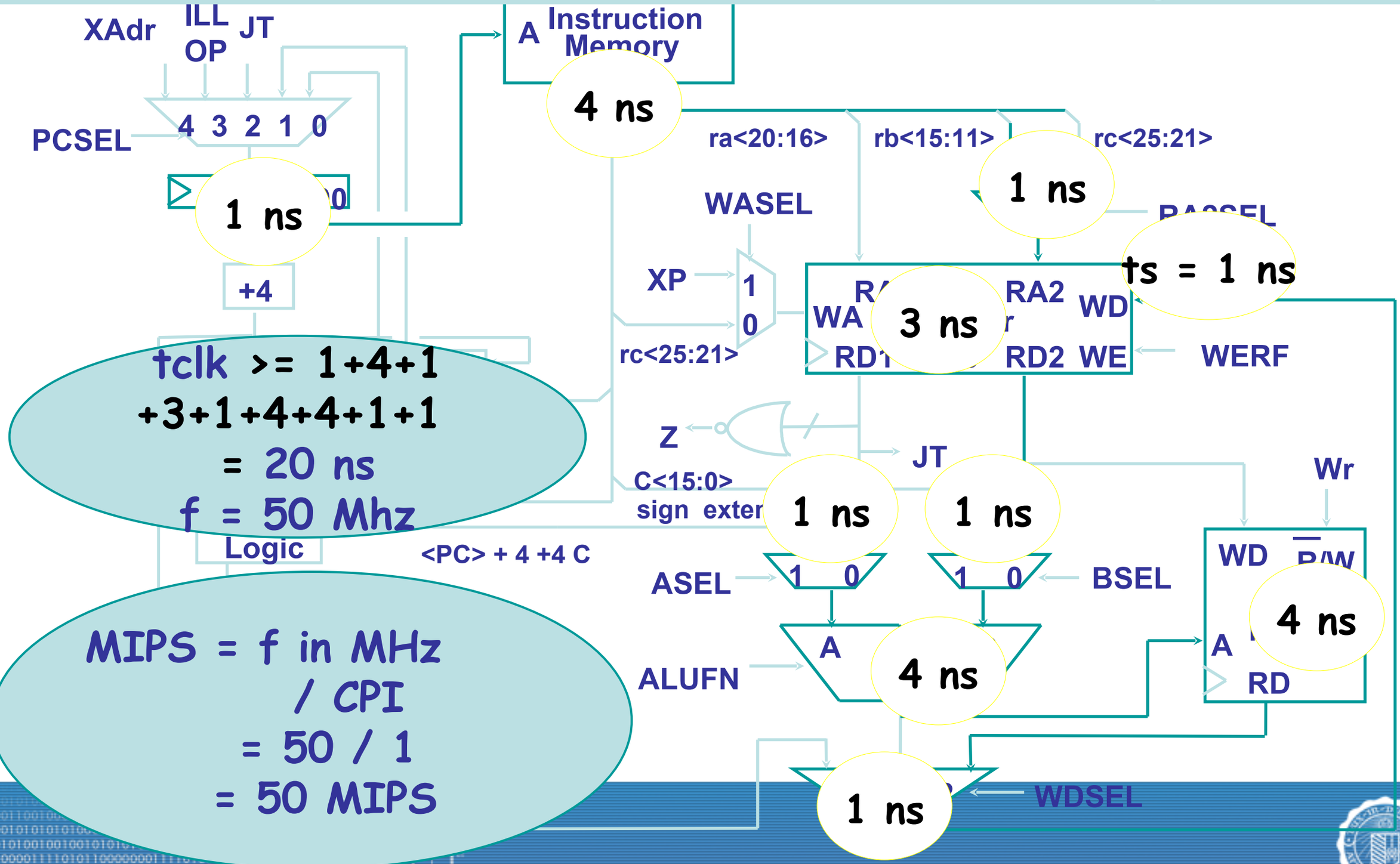
- Clocks Per Instruction (CPI)
  - Measure of *throughput.*
  - CPI = Number of clock cycles / Number of instructions finished in that time
- Frequency: $1 / t_{clk\text{-}min}$

CPU Performance



CPU Performance

Longest delay here is: PC -> Imem -> Iword -> RA1,RA2 -> RegFile -> RD1, RD2 -> ALU -> EA -> DMem -> Dword -> RegFileWD



DISCS

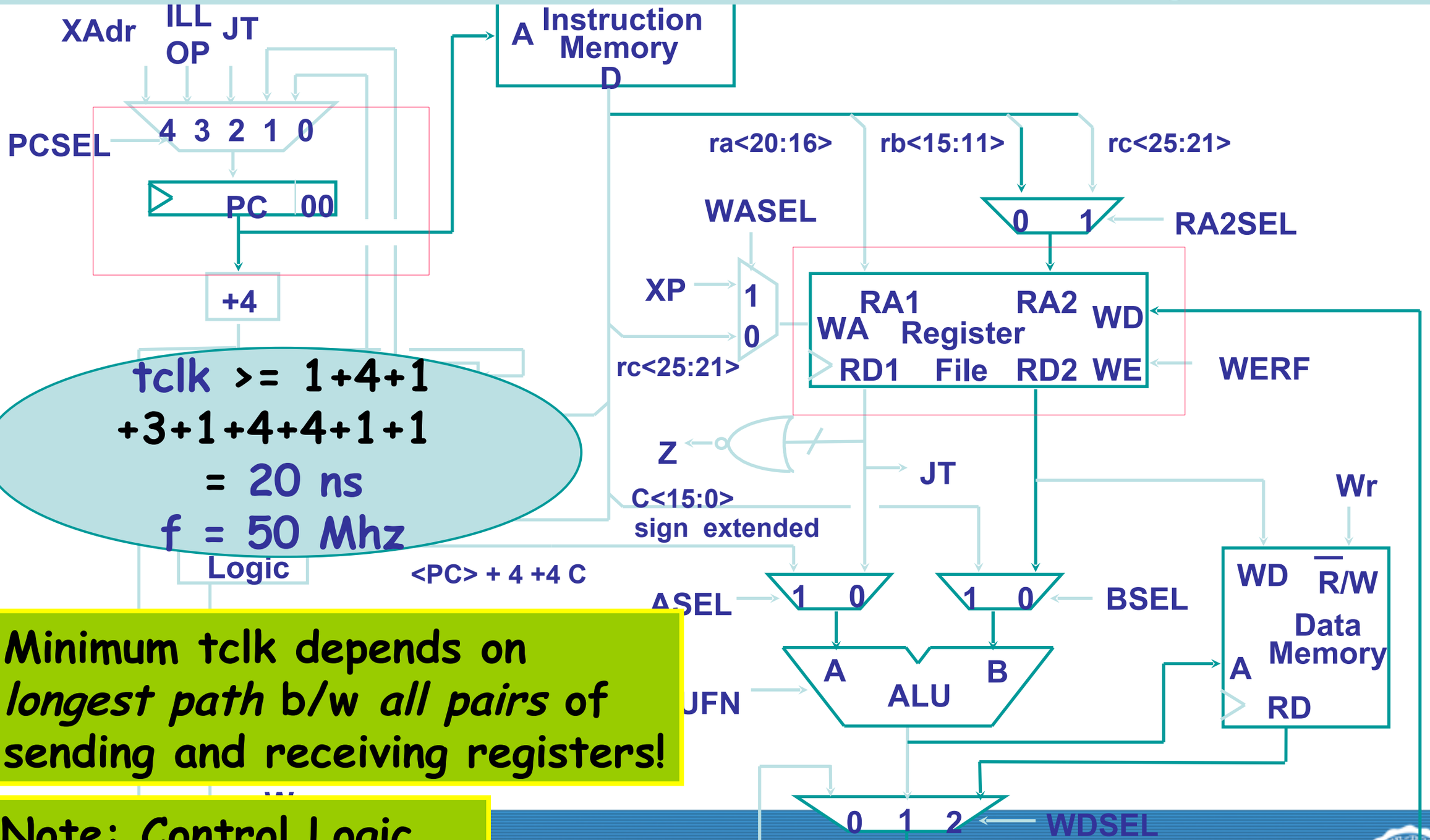
CPU Performance

$$\text{MIPS} = \frac{\text{Frequency in MHz}}{\text{Clocks Per Instruction (CPI)}}$$

- How can we increase MIPS?
  - Decrease CPI
    - Instruction set simplicity reduces CPI to 1.0!
      - CISC machines take several cycles per instruction, unlike RISC Beta.
    - CPI below 1.0 is possible with multiple instruction issue machines.
  - Increase Frequency
    - Frequency limited by *longest combinational path* between register outputs to register inputs.
    - Solution: Pipelining!

CPU Performance

Longest delay here is: PC -> Imem -> Iword -> RA1,RA2 -> RegFile -> RD1, RD2 -> ALU -> EA -> DMem -> Dword -> RegFileWD



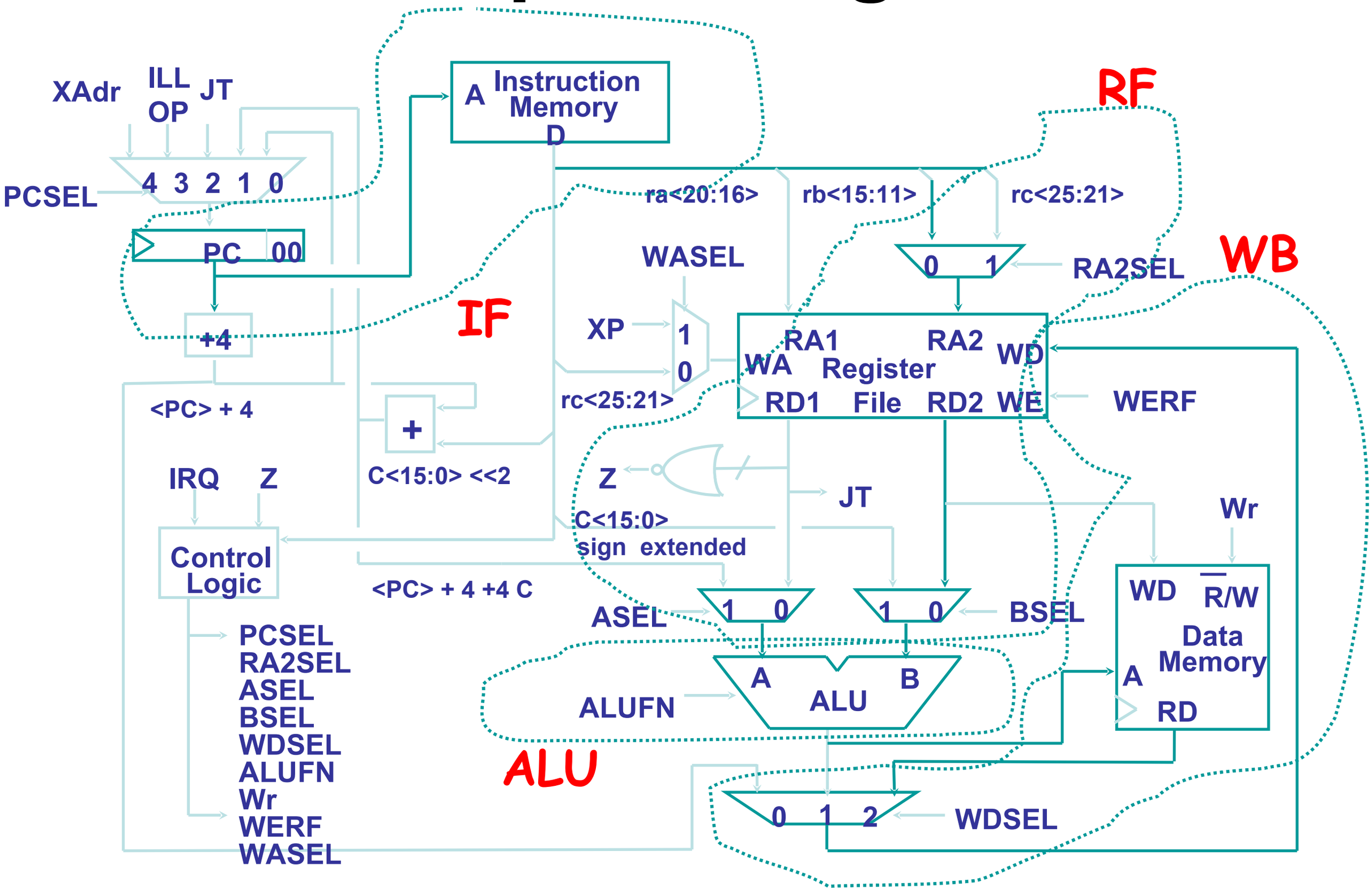
tclk >= 1+4+1
+3+1+4+4+1+1
= 20 ns
f = 50 Mhz

Minimum tclk depends on *longest path* b/w *all pairs* of sending and receiving registers!

Note: Control Logic can add delay too!

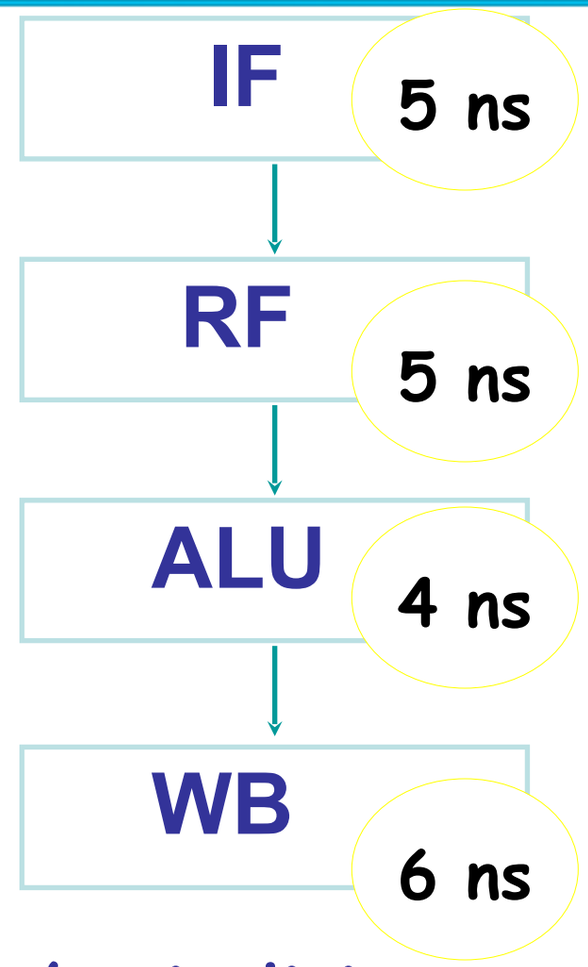
DISCS

Datapath Stages



Pipelining the Datapath

- **Goal**: Maintain (nearly) 1.0 CPI, but increase clock speed.
- **Approach**: Structure processor as 4-stage pipeline:
  - **Instruction Fetch**: Maintains **PC**. Fetches one instruction per cycle.
  - **Register File**: Reads source operands from register file.
  - **ALU**: Performs indicated operation.
  - **Write-Back**: Writes result back into register file.



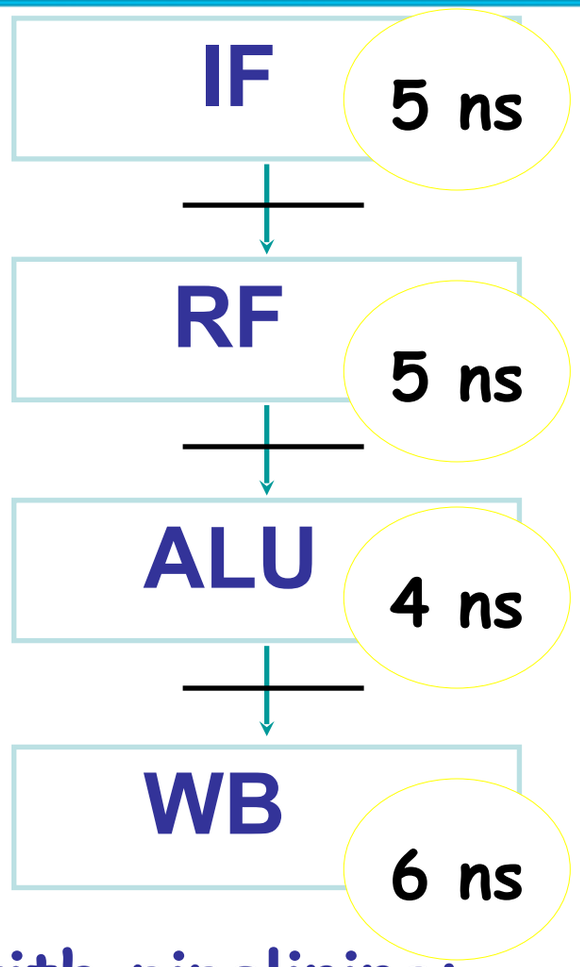
For now: Assume no control logic delay

w/o pipelining:

tclk-min = 20 ns
fmax = 50 Mhz
MIPS = 50 MIPS

Pipelining the Datapath

- **Goal**: Maintain (nearly) 1.0 CPI, but increase clock speed.
- **Approach**: Structure processor as 4-stage pipeline:
  - **Instruction Fetch**: Maintains **PC**. Fetches one instruction per cycle.
  - **Register File**: Reads source operands from register file.
  - **ALU**: Performs indicated operation.
  - **Write-Back**: Writes result back into register file.



Also assume ideal registers

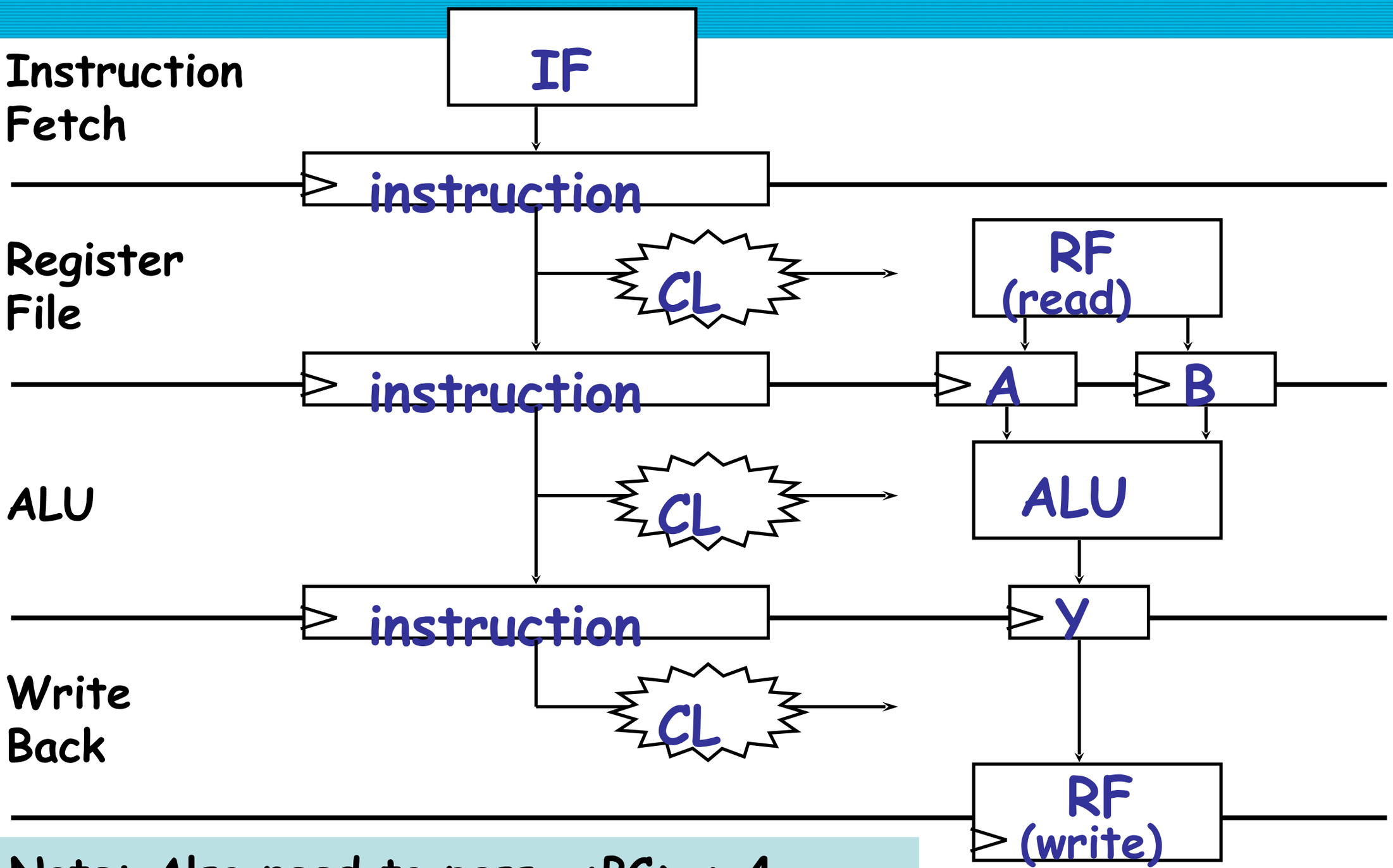
with pipelining:

tclk-min = 6 ns
fmax = 166 MHz
MIPS = 166 MIPS

Did You Know?

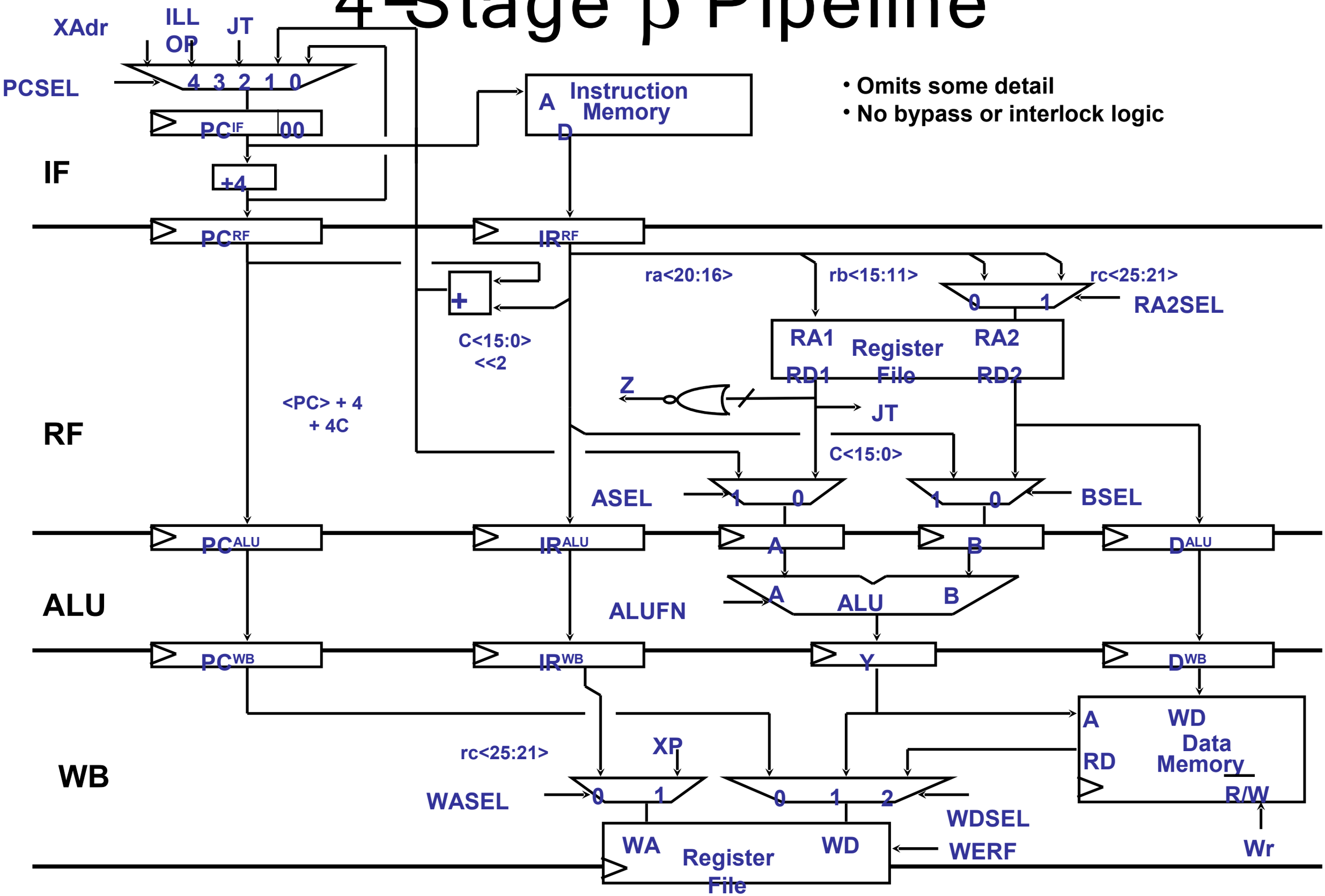
- **Factoids:**
  - Light travels roughly 1 ft. / ns.
  - Current processors run at around 3.0 GHz,
    - which means tclk = 0.33 ns!
- **Thus:**
  - Light only gets to travel *4 inches per instruction*!
    - And that's assuming a CPI of 1.0!
    - Many of today's processors can do CPIs of 1/2 or 1/4 by running several instructions at a time!

Sketch of 4-Stage Pipeline



Note: Also need to pass <PC> + 4 through the stages for branch and jump instructions!

4-Stage β Pipeline



- Omits some detail
- No bypass or interlock logic

4-Pipeline Execution

- Consider a sequence of instructions:

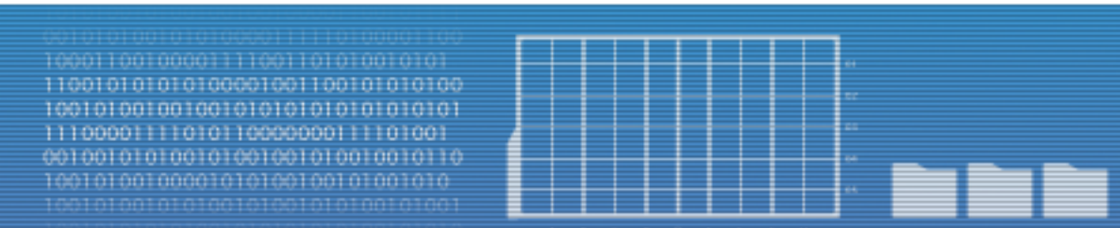
0x100: ADDC(r1, 1, r2)

0x104: SUBC(r1, 1, r3)

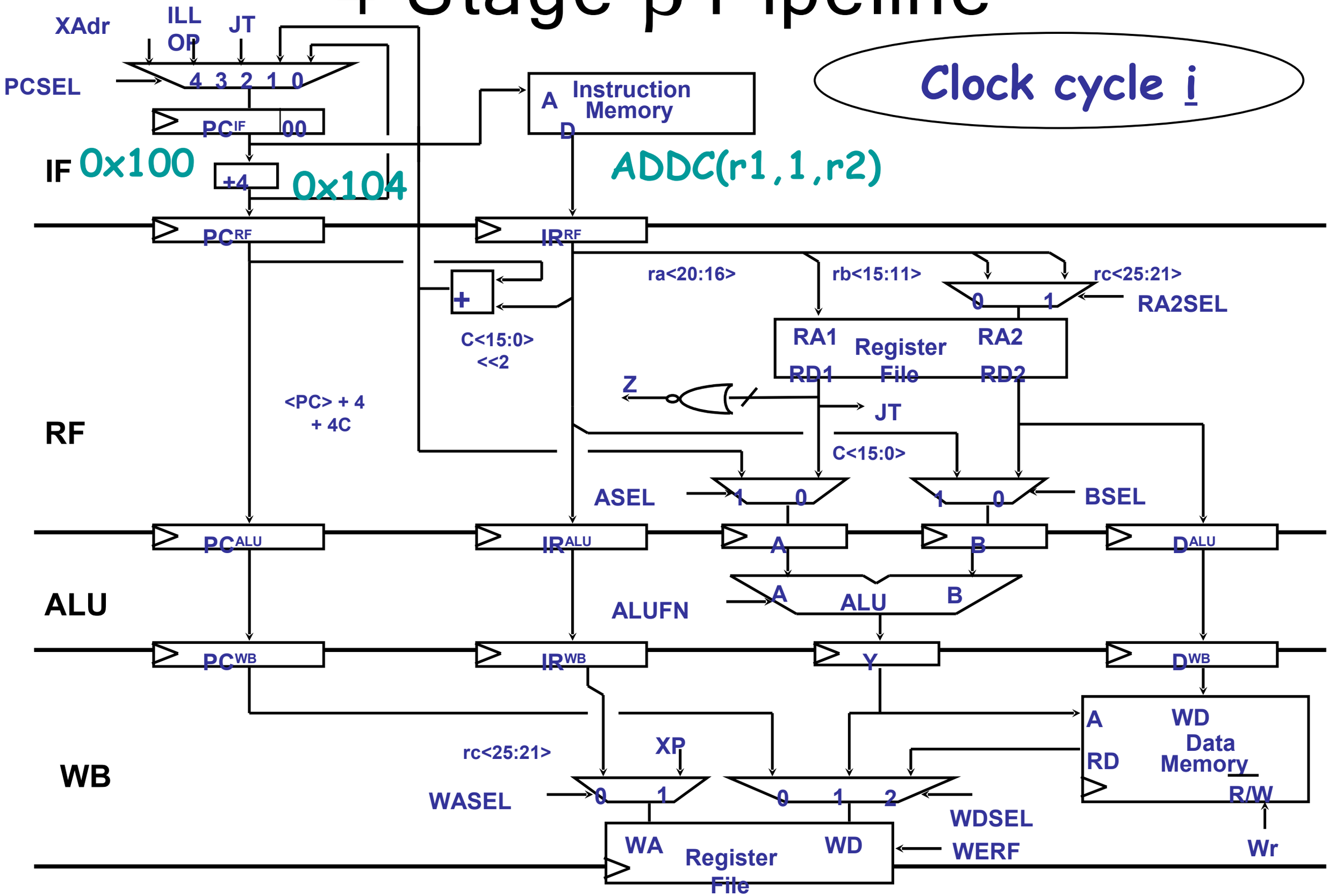
0x108: XOR(r1, r5, r1)

0x10C: MUL(r2, r5, r0)

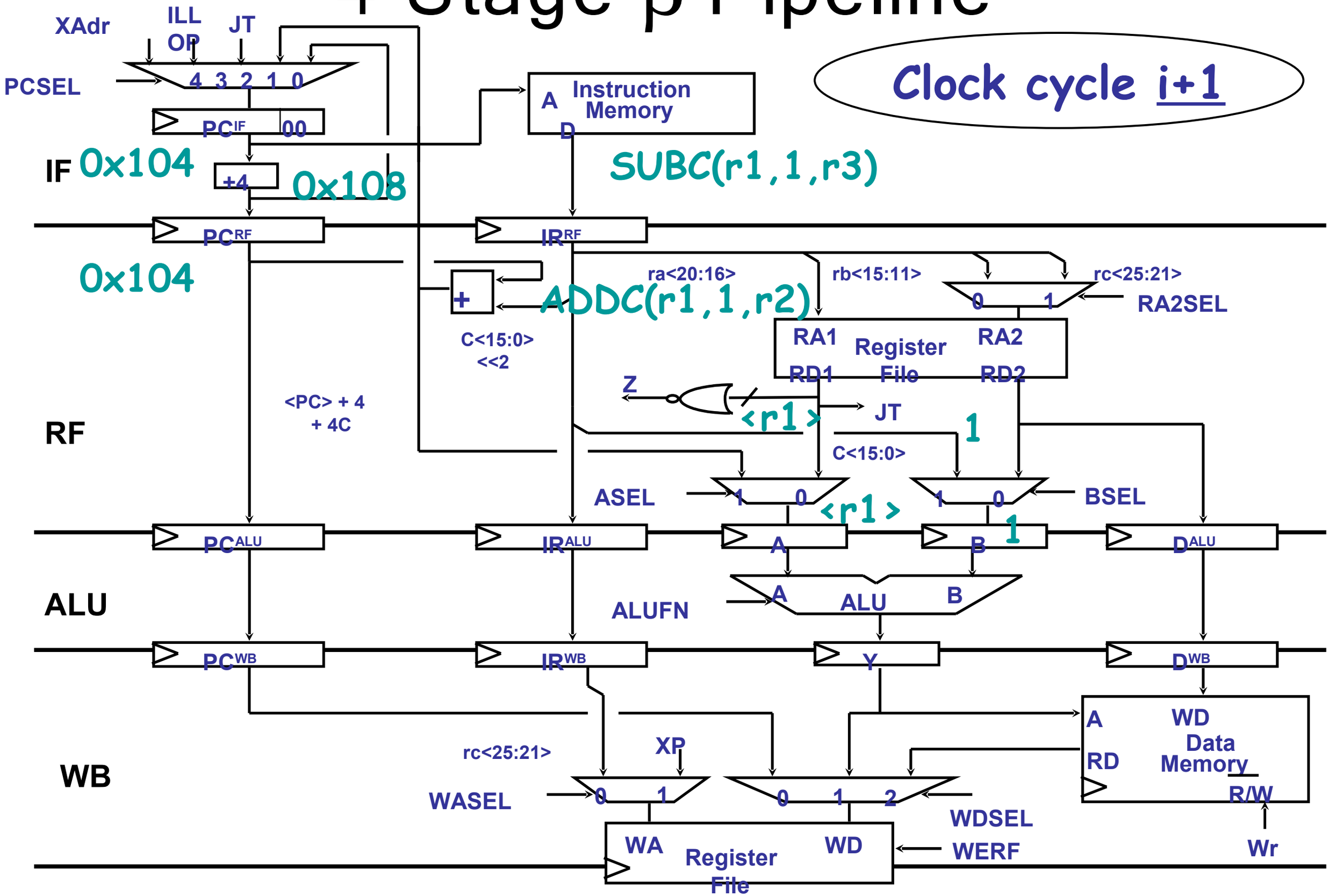
to be executed on the 4-stage pipeline...



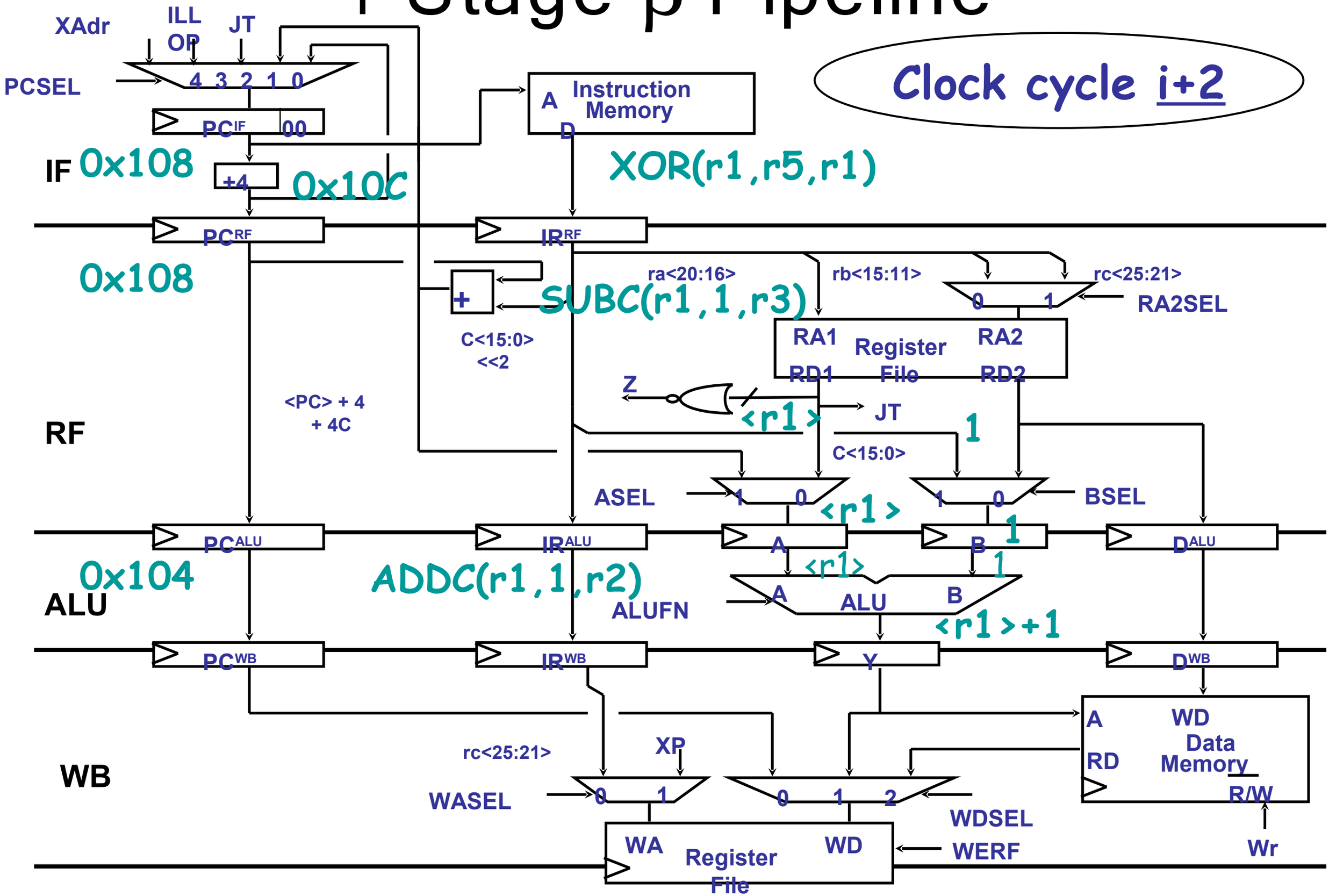
4-Stage β Pipeline



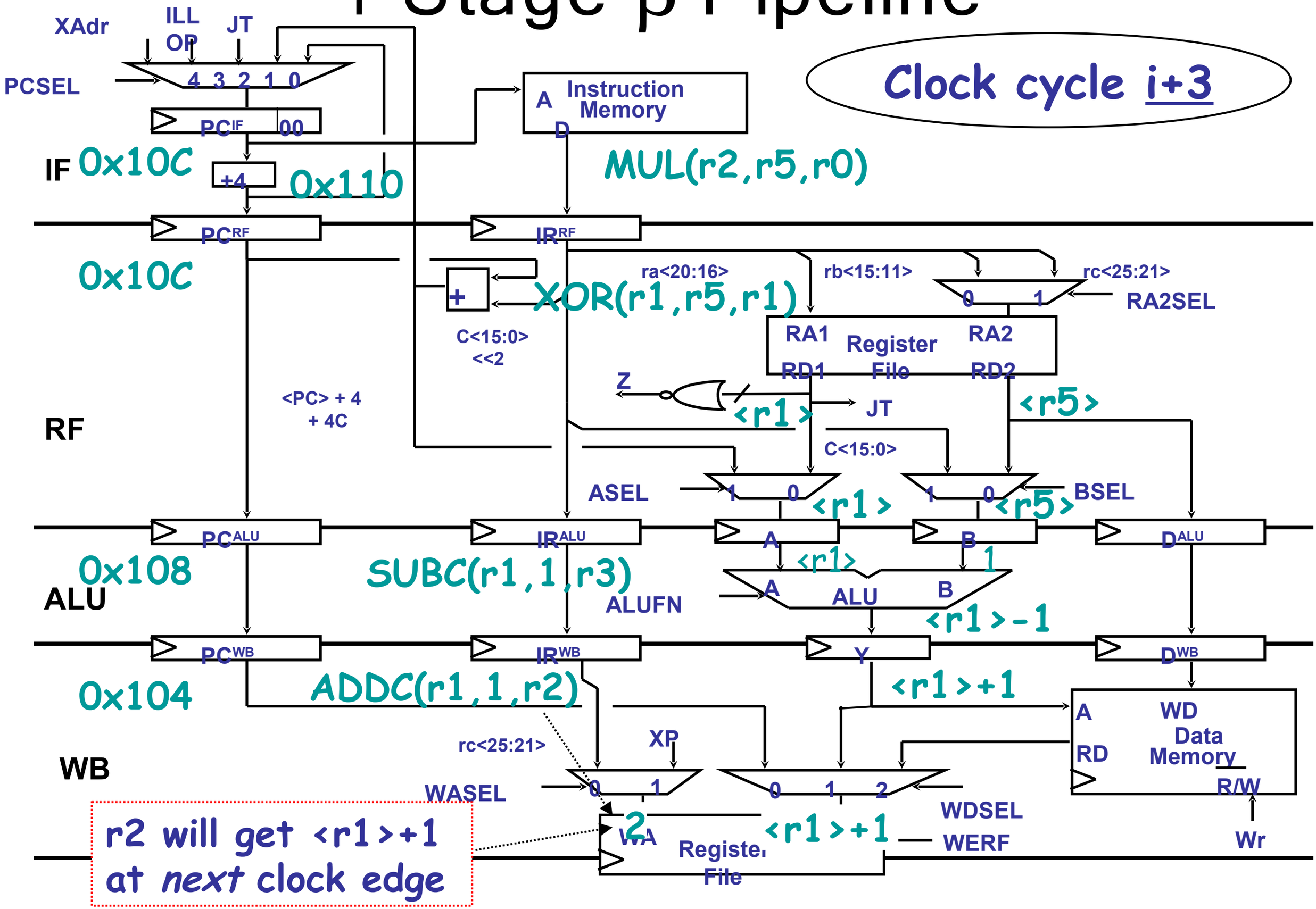
4-Stage β Pipeline



4-Stage β Pipeline



4-Stage β Pipeline



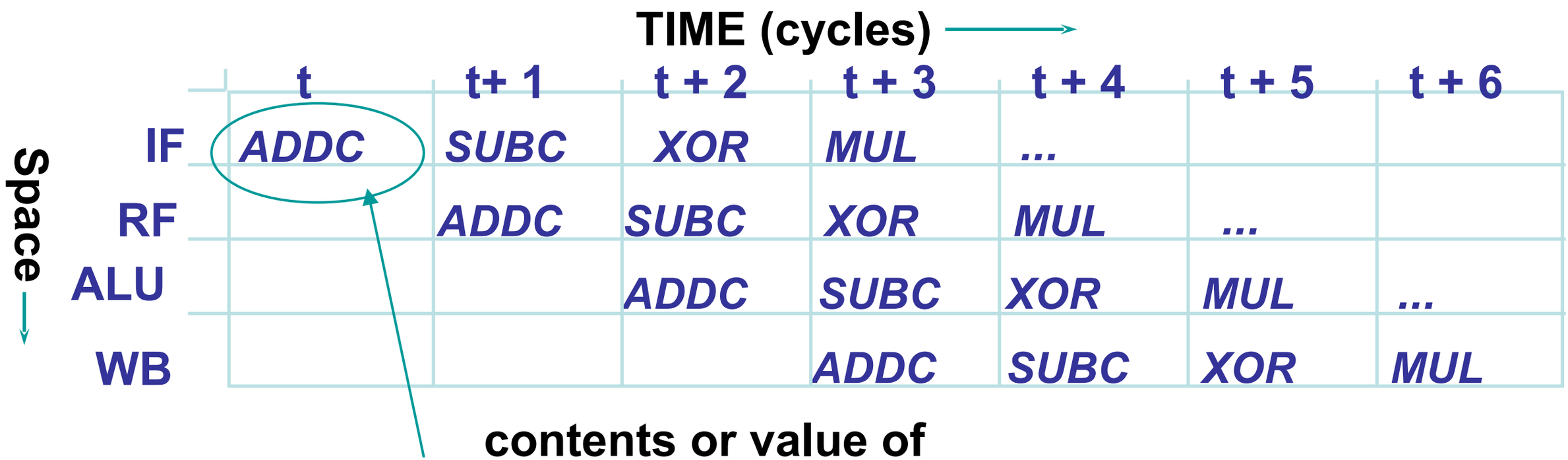
4-Stage β Pipeline

Clock cycle i+4

XAdr
ILL OP
JT
PCSEL
4 3 2 1 0
PC^IF
00
A
Instruction Memory
D
IF
0x110
+4
0x114
...
PC^RF
IR^RF
0x110
ra<20:16>
MUL(r2,r5,r0)
rb<15:11>
rc<25:21>
0
1
RA2SEL
RA1
Register File
RA2
RD1
RD2
this is the <r2> written by the ADD at the clock edge after clk i+3
Z
<r2>
JT
<r5>
RF
C<15:0>
ASEL
1
0
<r2>
1
0
<r5>
BSEL
PC^ALU
IR^ALU
A
B
D^ALU
0x10C
XOR(r1,r5,r1)
<r1>
<r5>
ALU
ALUFN
A
ALU
B
<r1>xor<r5>
PC^WB
IR^WB
Y
D^WB
0x108
SUBC(r1,1,r3)
<r1>-1
A
WD
Data Memory
RD
R/W
rc<25:21>
XP
WB
WASEL
0
1
0
1
2
3
<r1>-1
WDSEL
WA
Register File
WD
WERF
Wr
r3 will get <r1>-1 at *next* clock edge

Pipeline Timing Diagram

- Space-Time Diagram
  - Shows contents/values of parts/signals across time.
  - Shows “What is where?” at time t.
  - Can also be used to show register values (see later slides).



Another Way

- Instruction-Time Diagram
  - Shows “Where am I?” at time t.
  - Used in some textbooks and tutorials.

TIME (cycles) →

Instructions ↓	t	t+ 1	t + 2	t + 3	t + 4	t + 5	t + 6
ADDC	*IF*	*RF*	*ALU*	*WB*	*...*		
SUBC		*IF*	*RF*	*ALU*	*WB*	*...*	
XOR			*IF*	*RF*	*ALU*	*WB*	*...*
MULC				*IF*	*RF*	*ALU*	*WB*

shows which stage ADDC instruction is at time t

Is there a Price to be Paid?

suppose: r1=7, r2=9, r3=10

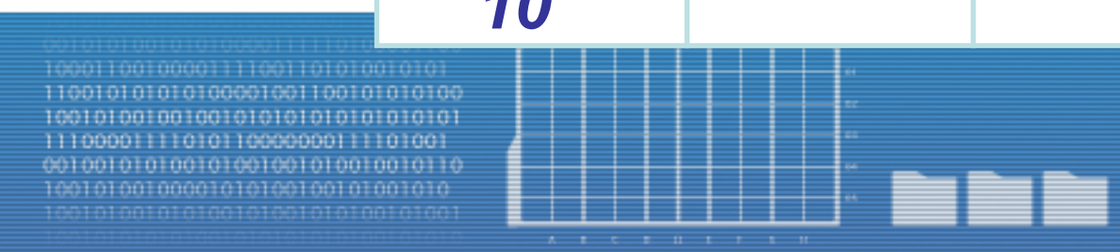
0x100: ADDC(r1, 1, r2)
0x104: SUBC(r1, 1, r3)
0x108: XOR(r1, r5, r1)
0x10C: MUL(r2, r5, r0)

- ADDC reads from r1, writes to r2, but…

TIME (cycles) →

Space ↓

	t	t+ 1	t + 2	t + 3	t + 4	t + 5	t + 6
IF	*ADDC*	*SUBC*	*XOR*	*MUL*	*...*		
RF		*ADDC*	*SUBC*	*XOR*	*MUL*	*...*	
ALU			*ADDC*	*SUBC*	*XOR*	*MUL*	*...*
WB				*ADDC*	*SUBC*	*XOR*	*MUL*
PC_{IF}	*0x100*	*0x104*	*0x108*	*0x10C*	*0x110*	*...*	
R1	*7*						
R2	*9*						
R3	*10*						



DISCS

4-Stage β Pipeline

Clock cycle i+3

XAdr
ILL OP
JT
PCSEL
4 3 2 1 0
PC^IF
00
IF
0x10C
+4
0x110
A
Instruction Memory
D
MUL(r2,r5,r0)
PC^RF
IR^RF
0x10C
ra<20:16>
rb<15:11>
rc<25:21>
+
XOR(r1,r5,r1)
0
1
RA2SEL
C<15:0>
<<2
RA1
Register File
RA2
RD1
RD2
Z
<r1>
JT
<r5>
RF
<PC> + 4
+ 4C
C<15:0>
ASEL
1
0
<r1>
1
0
<r5>
BSEL
PC^ALU
IR^ALU
A
B
D^ALU
0x108
SUBC(r1,1,r3)
<r1>
1
A
ALU
B
ALU
ALUFN
<r1>-1
PC^WB
IR^WB
Y
D^WB
0x104
ADDC(r1,1,r2)
<r1>+1
A
WD
Data Memory
RD
R/W
rc<25:21>
XP
WB
WASEL
0
1
0
1
2
WDSEL
2
WA
<r1>+1
WERF
Wr
Register File
r2 will get <r1>+1
at *next* clock edge
after ADDC gets here!

4-Stage β Pipeline

Clock cycle i+4

XAdr
ILL OP
JT
PCSEL
4 3 2 1 0
PC^IF
00
IF
0x110
+4
0x114
A
Instruction Memory
D
...
PC^RF
IR^RF
0x110
ra<20:16>
rb<15:11>
rc<25:21>
MUL(r2,r5,r0)
0
1
RA2SEL
RA1
Register File
RA2
RD1
RD2
Z
<r2>
JT
<r5>
RF
this is the <r2> written by the ADD at the clock edge after clk i+3
C<15:0>
ASEL
1
0
<r2>
1
0
<r5>
BSEL
PC^ALU
IR^ALU
A
B
D^ALU
0x10C
XOR(r1,r5,r1)
ALUFN
<r1>
<r5>
A
ALU
B
ALU
<r1>xor<r5>
PC^WB
IR^WB
Y
D^WB
0x108
SUBC(r1,1,r3)
<r1>-1
A
WD
Data Memory
RD
R/W
rc<25:21>
XP
WASEL
0
1
0
1
2
WDSEL
WA
Register File
WD
WERF
Wr
WB

Register Write-Back is Delayed

- Not only for pipeline stages, but for register and wire values as well!
- Looks OK for now, since MUL reads updated R2 value at cycle i+4.

suppose r1=7,r2=9,r3=10,r5=3

```
0x100: ADDC(r1, 1, r2)
0x104: SUBC(r1, 1, r3)
0x108: XOR(r1, r5, r1)
0x10C: MUL(r2, r5, r0)
```

TIME (cycles)

Space →

	t	t+ 1	t + 2	t + 3	t + 4	t + 5	t + 6
IF	*ADDC*	*SUBC*	*XOR*	*MUL*	*...*		
RF		*ADDC*	*SUBC*	*XOR*	*MUL*	*...*	
ALU			*ADDC*	*SUBC*	*XOR*	*MUL*	*...*
WB				*ADDC*	*SUBC*	*XOR*	*MUL*
PC_{IF}	*0x100*	*0x104*	*0x108*	*0x10C*	*0x110*	*...*	
R1	*7*	*7*	*7*	*7*	*7*	*7*	
R2	*9*	*9*	*9*	*9*	*8*	*8*	
R3	*10*	*10*	*10*	*10*	*10*	*6*	

What if MUL was Earlier?

- MUL gets 9.
  - A stale (and incorrect) value!

```
0x100: ADDC(r1, 1, r2)
0x104: SUBC(r1, 1, r3)
0x108: MUL(r2, r5, r0)
0x10C: XOR(r1, r5, r1)
```

TIME (cycles) →

Space ↓

	t	t+ 1	t + 2	t + 3	t + 4	t + 5	t + 6
IF	ADDC	SUBC	MUL	XOR	...		
RF		ADDC	SUBC	MUL	XOR	...	
ALU			ADDC	SUBC	MUL	XOR	...
WB				ADDC	SUBC	MUL	XOR
PC_{IF}	0x100	0x104	0x108	0x10C	0x110	...	
R1	7	7	7	7	7	7	
R2	9	9	9	9	8	8	
R3	10	10	10	10	10	6	

DISCS

Write-Back Delay Slots

- Problem: Registers are not written back right away, so if later instructions try to read too early, they get “stale” values.
- Possible solution:
- Insert NOPs
  - ADD(R31,R31,R31) | does absolutely nothing!
- Reorder Code
  - Move non-dependent code into delay slot.

Inserting NOPs

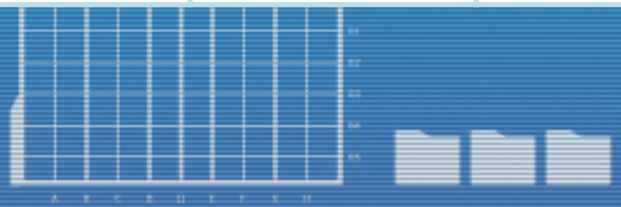
- How Many?

```
0x100: ADDC(r1, 1, r2)
0x104: SUBC(r1, 1, r3)
0x108: MUL(r2, r5, r0)
0x10C: XOR(r1, r5, r1)
```

TIME (cycles) →

Space ↓

	t	t+ 1	t + 2	t + 3	t + 4	t + 5	t + 6
IF	ADDC	SUBC					
RF		ADDC	SUBC				
ALU			ADDC	SUBC			
WB				ADDC	SUBC		
PC_{IF}	0x100	0x104	0x108	0x10C	0x110	...	
R1	7	7	7	7	7	7	
R2	9	9	9	9	8	8	
R3	10	10	10	10	10	6	

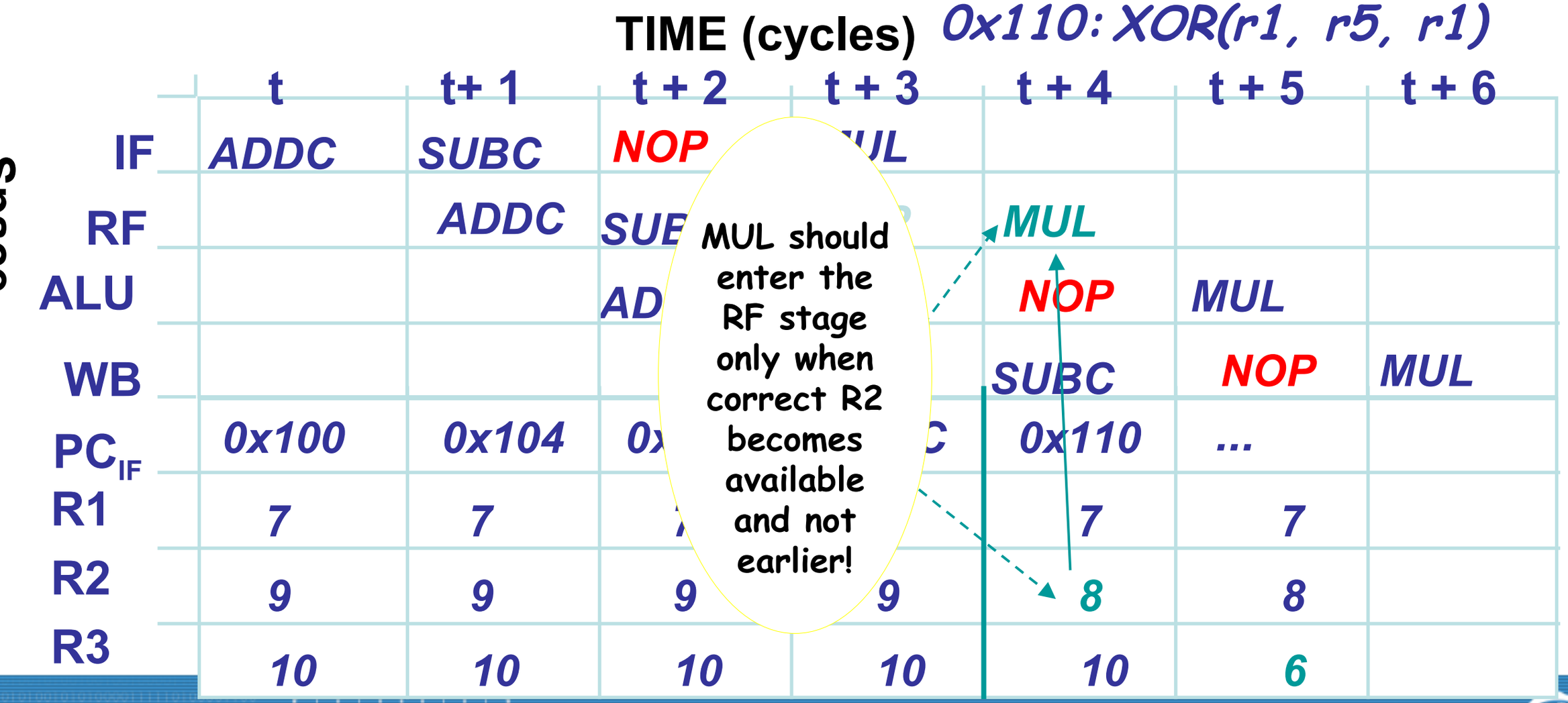


DISCS

Inserting NOPs

- Need to put reading instruction (MUL) in RF stage in the same cycle that correct values of regs become available (must be at least the 3rd instruction after ADDC, which writes to R2.)

```
0x100: ADDC(r1, 1, r2)
0x104: SUBC(r1, 1, r3)
0x108: NOP()
0x10C: MUL(r2, r5, r0)
0x110: XOR(r1, r5, r1)
```



Reordering Code

- NOPs waste cycles.
- Instead of NOPs, we can use other instructions.
  - But they must not depend on changing regs!

```
0x100: ADDC(r1, 1, r2)
0x104: SUBC(r1, 1, r3)
0x108: XOR(r1, r5, r1)
0x10C: MUL(r2, r5, r0)
```

TIME (cycles)

Space ↓	t	t + 1	t + 2	t + 3	t + 4	t + 5	t + 6
IF	ADDC		OR	MUL			
RF			BC	XOR	MUL		
ALU			DC	SUBC	XOR	MUL	
WB				ADDC	SUBC	XOR	MUL
PC_{IF}	0x100		0x108	0x10C	0x110	...	
R1	7	7	7	7	7	7	
R2	9	9	9	9	8	8	
R3	10	10	10	10	10	6	

It's OK to move XOR before MUL because <r1> and <r5> are correct in cycle t+3

Write-Back Delays

- Consider the sequence:

  ADD(r1, r2, r3)

  CMPLEC(r3, 5, r0)

  MUL(r1, r2, r4)

  SUB(r1, r2, r5)

	i	i + 1	i + 2	i + 3	i + 4	i + 5	i + 6
IF	*ADD*	*CMP*	*MUL*	*SUB*			
RF		*ADD*	*CMP*	*MUL*	*SUB*		
ALU			*ADD*	*CMP*	*MUL*	*SUB*	
WB				*ADD*	*CMP*	*MUL*	*SUB*

ADD writes new value in _r3_ during _____ cycle, which is available beginning of _____ cycle.

Value of _r3_ read by _CMP_ during _____ cycle.

CMP reads _____ value of _r3_.

Fix the problem:
- insert NOPs
- move code

What About Branching/Jumping?

- Consider a different sequence:

```
LOOP:  CMPLEC(r3, 100, r0)
       ADD(r1, r2, r3)
       SUB(r1, r2, r4)
       BNE(r0, LOOP)
       XOR(r9, r6, r3)
```

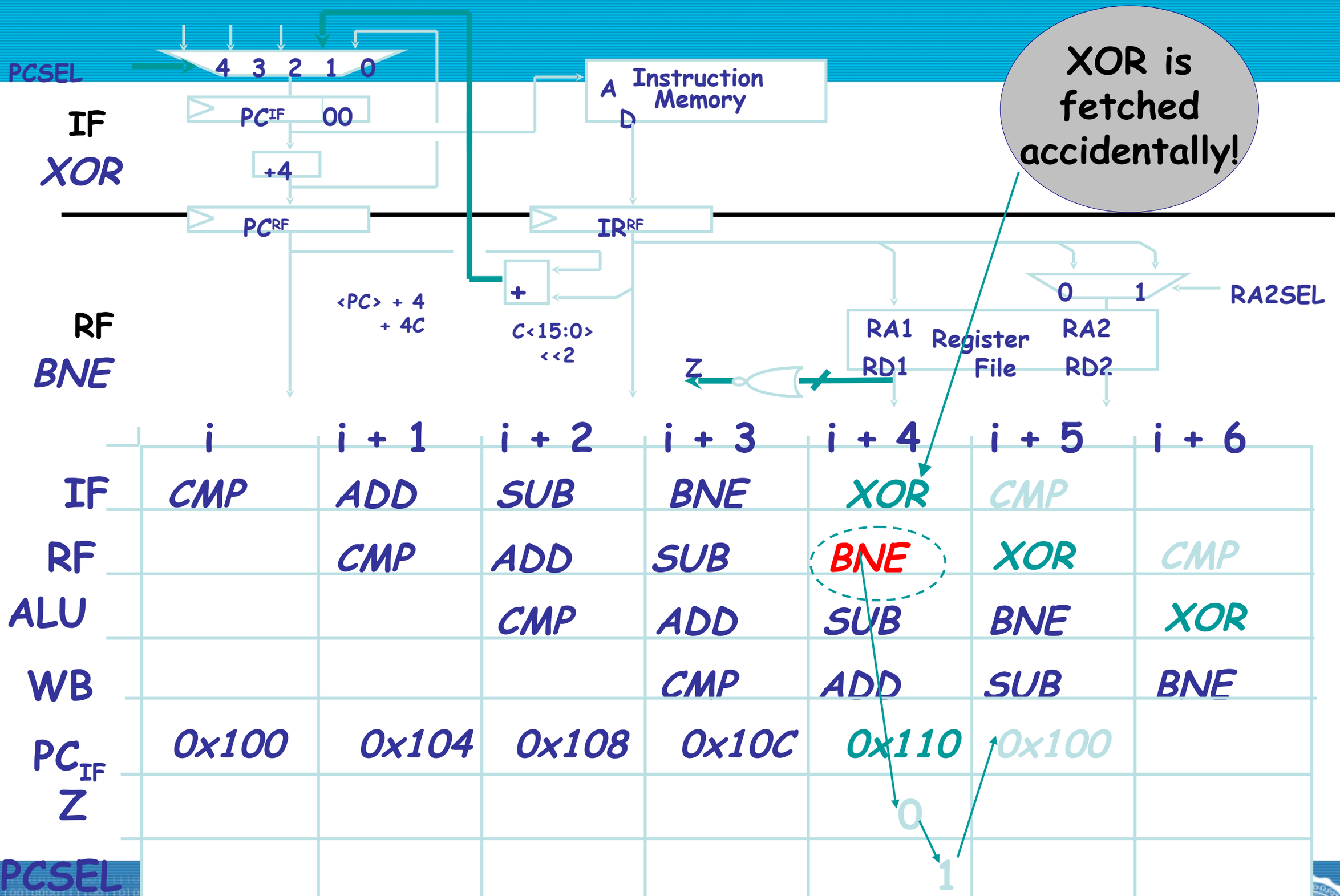
	i	i + 1	i + 2	i + 3	i + 4	i + 5	i + 6
IF	CMP	ADD	SUB	BNE	?		
RF		CMP	ADD	SUB	BNE		
ALU			CMP	ADD	SUB	BNE	
WB				CMP	ADD	SUB	BNE

What's wrong?

Think: What determines whether to branch or not?

(For JMP: How do we know where to jump?)

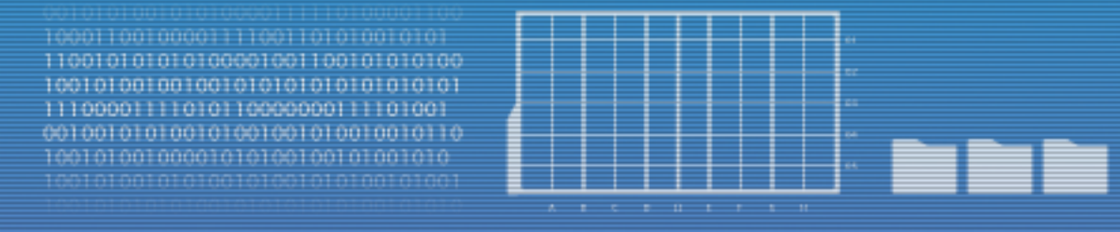
Branch Execution



Branch Delay Slots

- Problem: One (or more) instructions have been fetched by the time a branch can be decided.
- Software solution: Insert NOPs and/or reorder code!

	i	i + 1	i + 2	i + 3	i + 4	i + 5	i + 6
IF	CMP	ADD	SUB	BNE	XOR	CMP	
RF		CMP	ADD	SUB	BNE	XOR	CMP
ALU			CMP	ADD	SUB	BNE	XOR
WB				CMP	ADD	SUB	BNE
PC_{IF}	0x100	0x104	0x108	0x10C	0x110	0x100	
Z					0		
PCSEL					1		



Inserting NOPs

```
LOOP:  CMPLEC(r3, 100, r0)
       ADD(r1, r2, r3)
       SUB(r1, r2, r4)
       BNE(r0, LOOP)
       NOP()        <- "branch delay slot"
       XOR(r9, r6, r3)
```

- Add NOPs up to and including cycle where correct *target PC* is decided.
- What about reordering?

	i	i + 1	i + 2	i + 3	i + 4	i + 5	i + 6
IF	CMP	ADD	SUB	BNE	NOP	CMP	
RF		CMP	ADD	SUB	BNE	NOP	CMP
ALU			CMP	ADD	SUB	BNE	NOP
WB				CMP	ADD	SUB	BNE
PC_{IF}	0x100	0x104	0x108	0x10C	0x110	0x100	
Z					0		
PCSEL					1		

DISCS