COL216
Computer Architecture

Design a processor -
Single cycle & multi-cycle datapaths
15th February, 2018

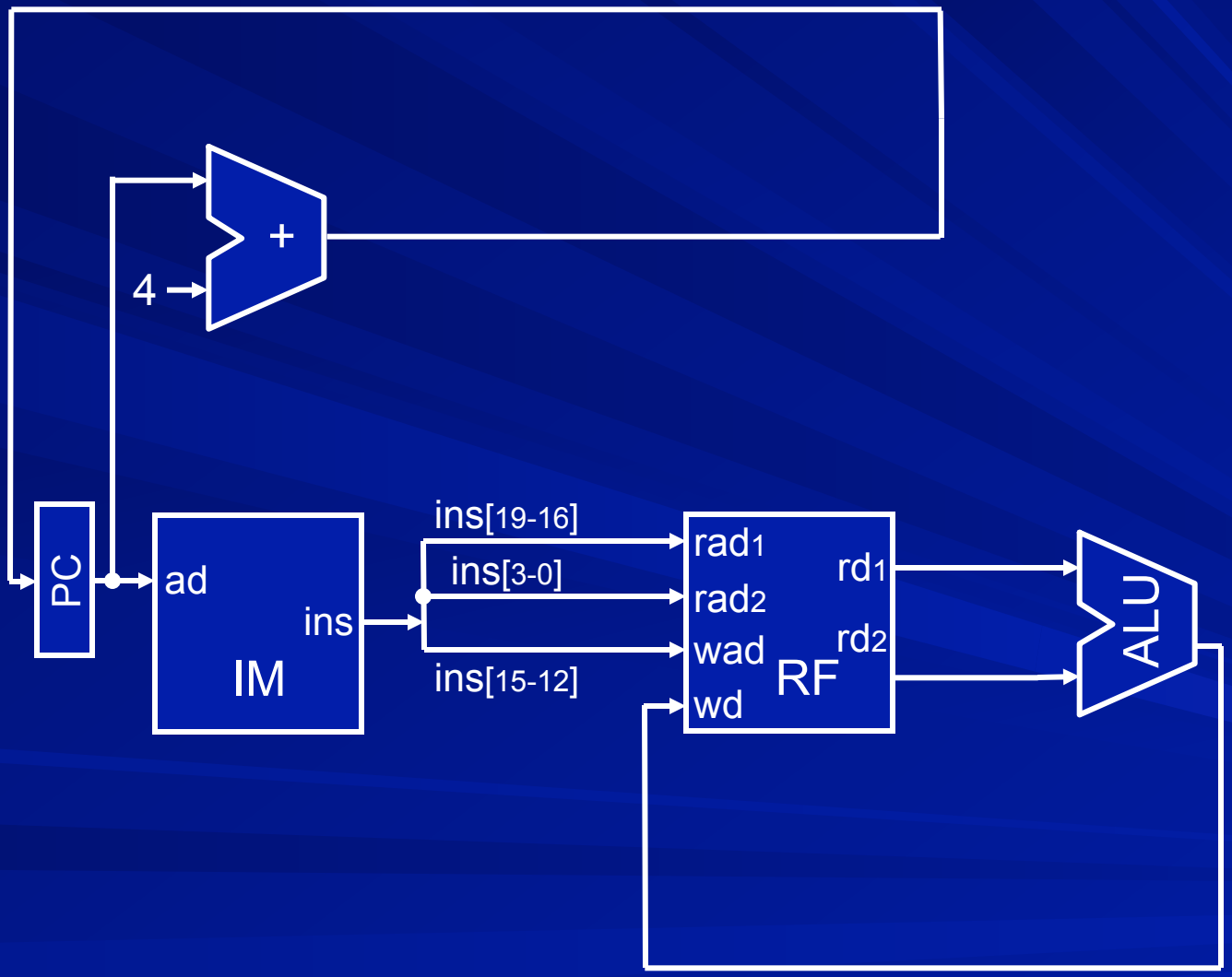
Assumptions

- DP with I = 0 only, i.e. Operand2 = reg
  - no shift/rotate (LSL #0)
- DT with I = 0 only, i.e. Operand2 = imm
  - also, P = 0, W = 0, B = 0
  - no H, SH, SB
- No MUL, MLA
- Only B,no BL
- Condition only with B
- No check for undef /non-impl instructions

Actions for DP instructions

- fetch instruction
- access the register file
- pass operands to ALU
- pass result to register file
- increment PC

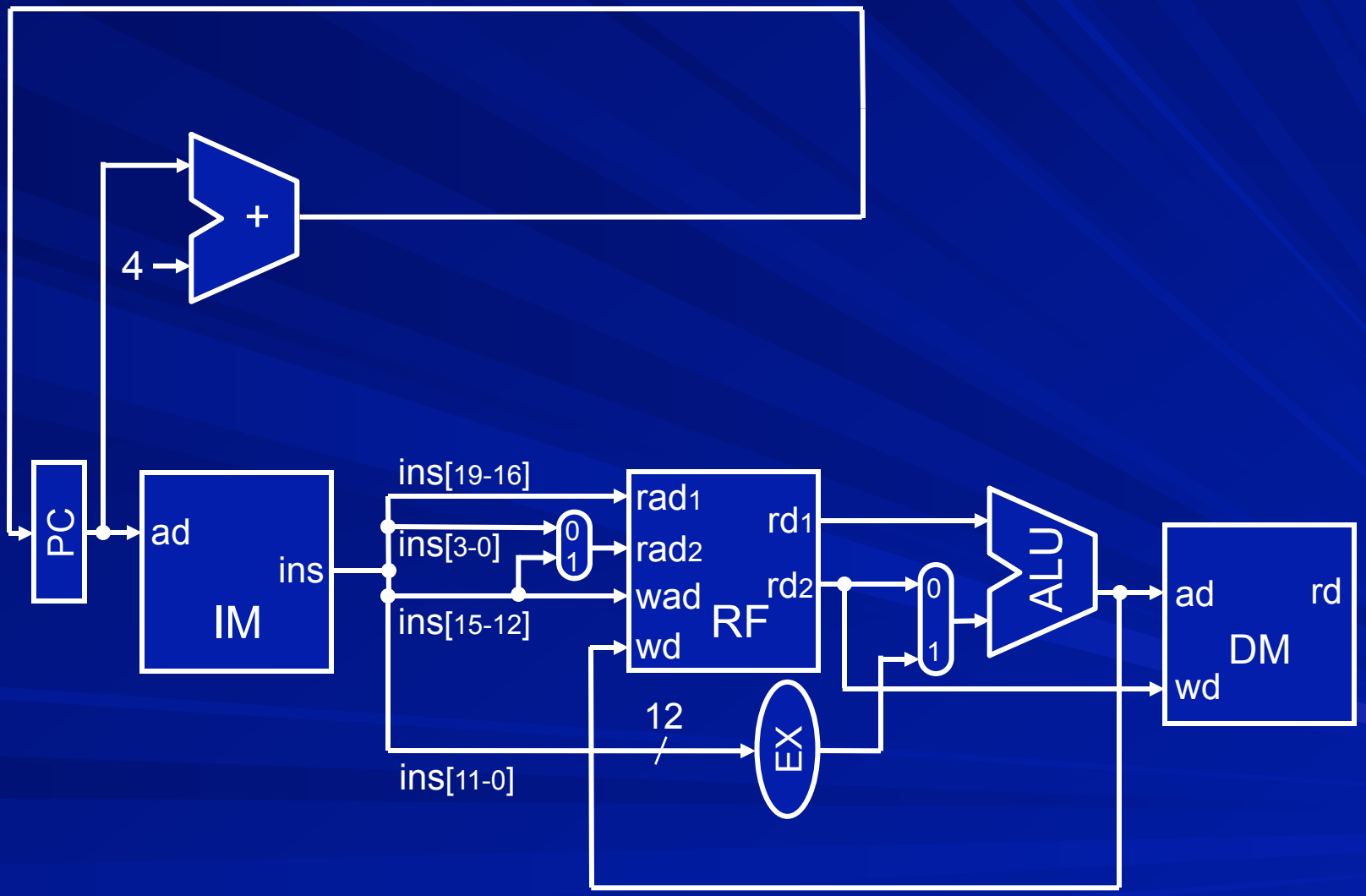
Datapath for DP instructions



Actions for str instructions

- fetch instruction
- access the register file
- compute address in ALU
- write data into memory
- increment PC

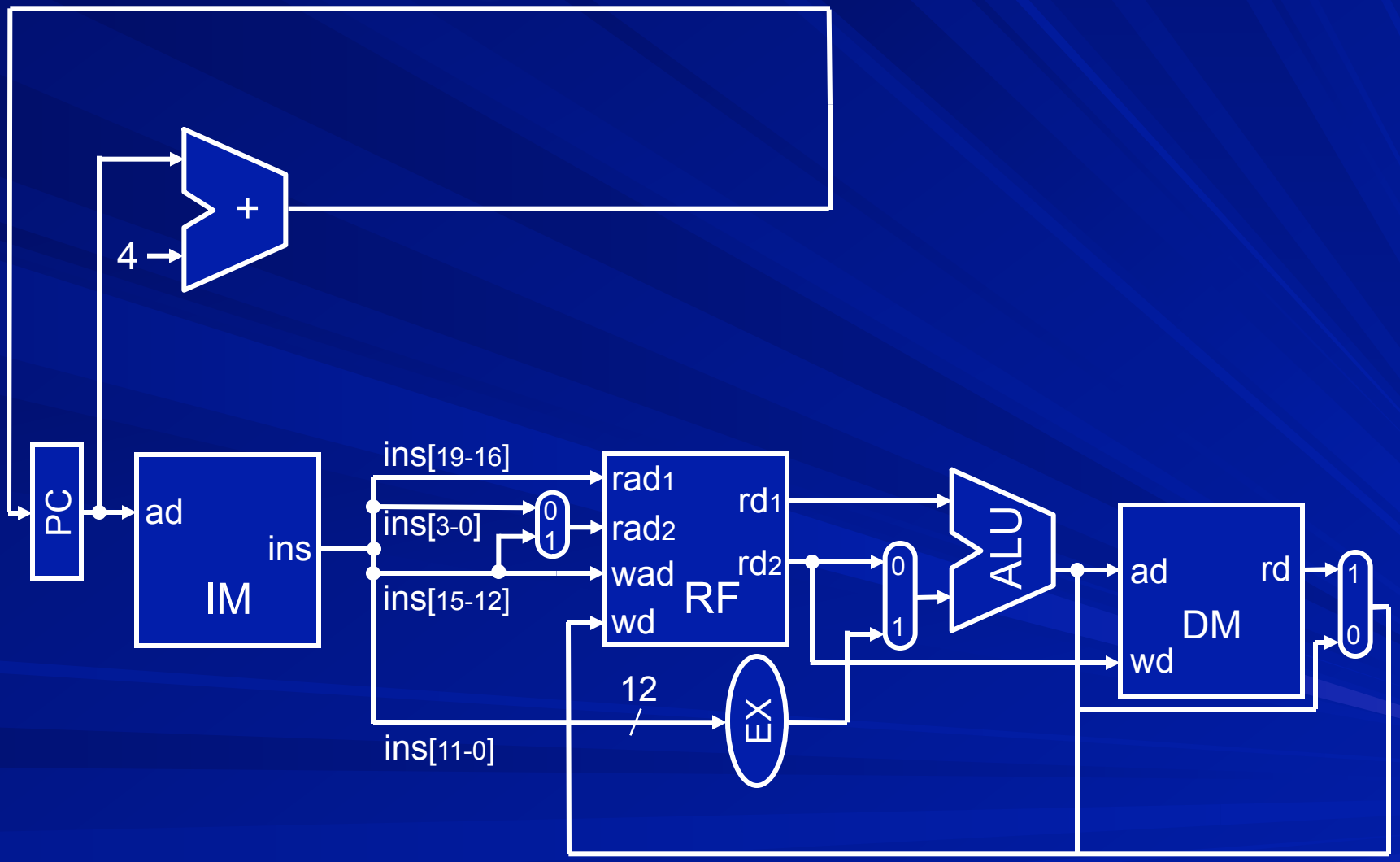
Adding “str” instruction



Actions for ldr instructions

- fetch instruction
- access the register file
- compute address in ALU
- read data from memory
- put data in register file
- increment PC

Adding “ldr” instruction

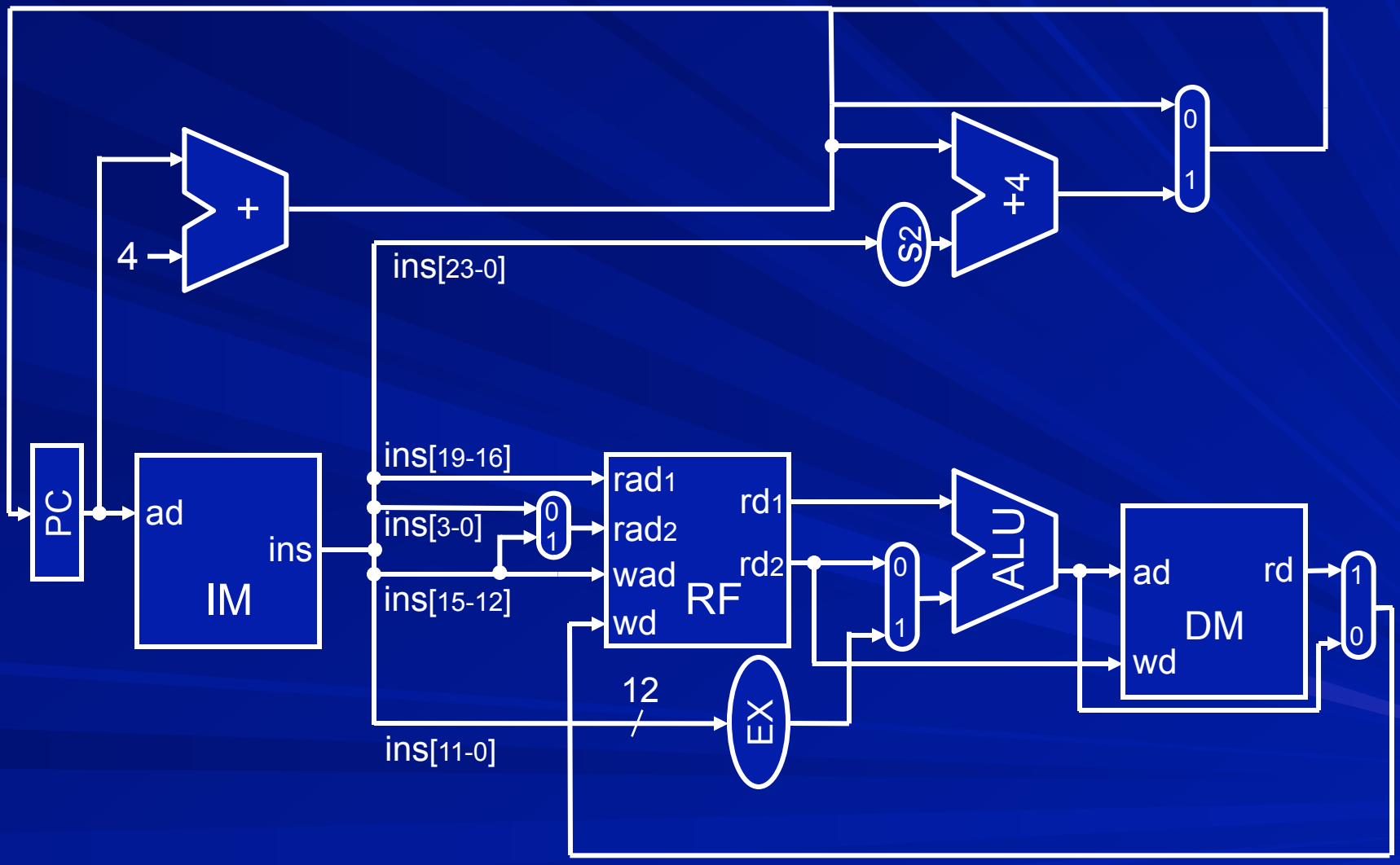


Actions for branch instruction

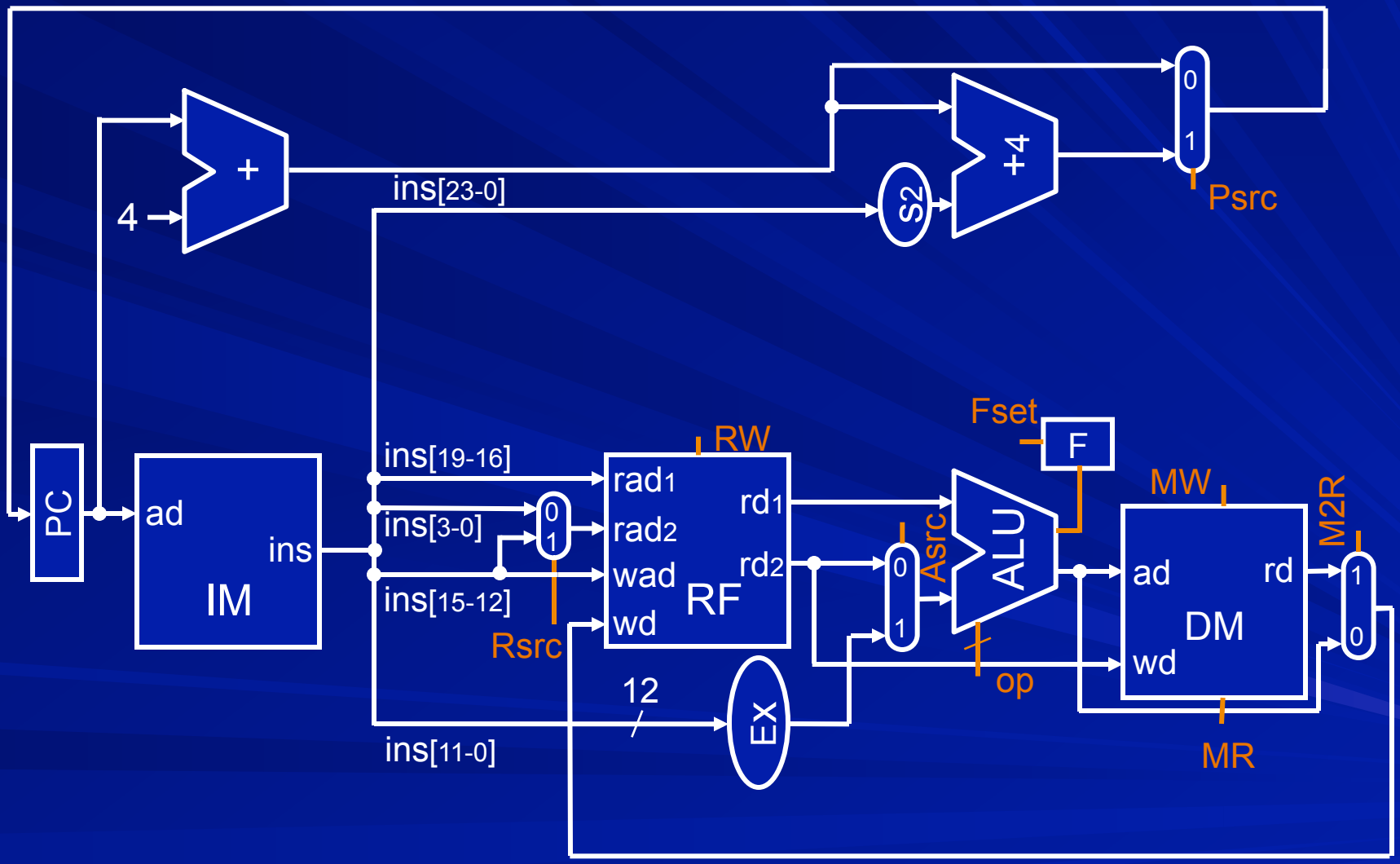
- fetch instruction

- compute target address
- transfer address to pc

Adding “b” instruction



Datapath + control signals



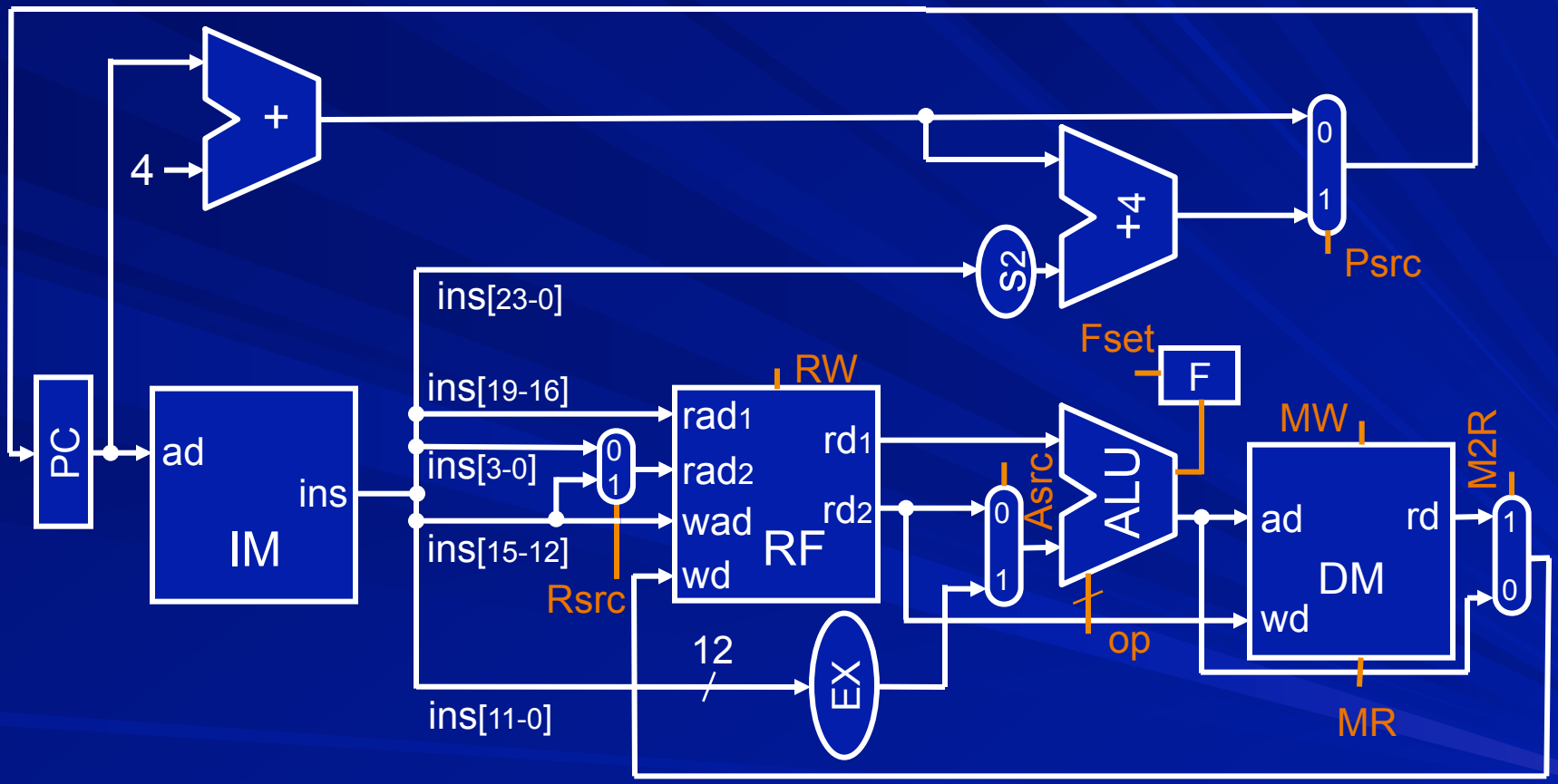
Problems with single cycle design

- Slowest instruction pulls down the clock frequency
- Resource utilization is poor
- There are some instructions which are impossible to be implemented in this manner

Analyzing performance

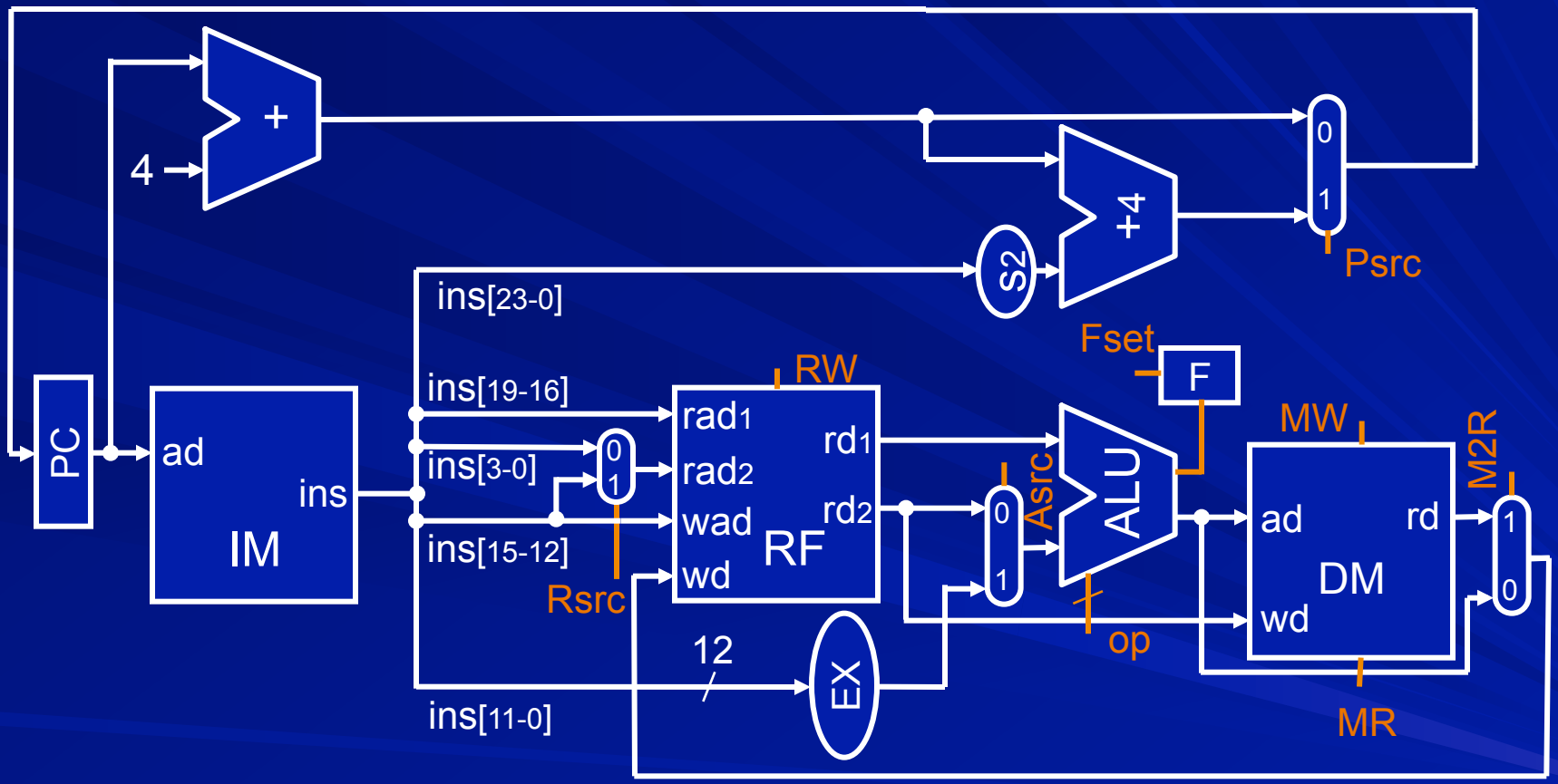
	Component delays
Register	0
Adder	t_+
ALU	t_A
Multiplexer	0
Register file	t_R
Program memory	t_I
Data memory	t_M
Bit manipulation components	0

Delay for {add, sub, and, orr, …}



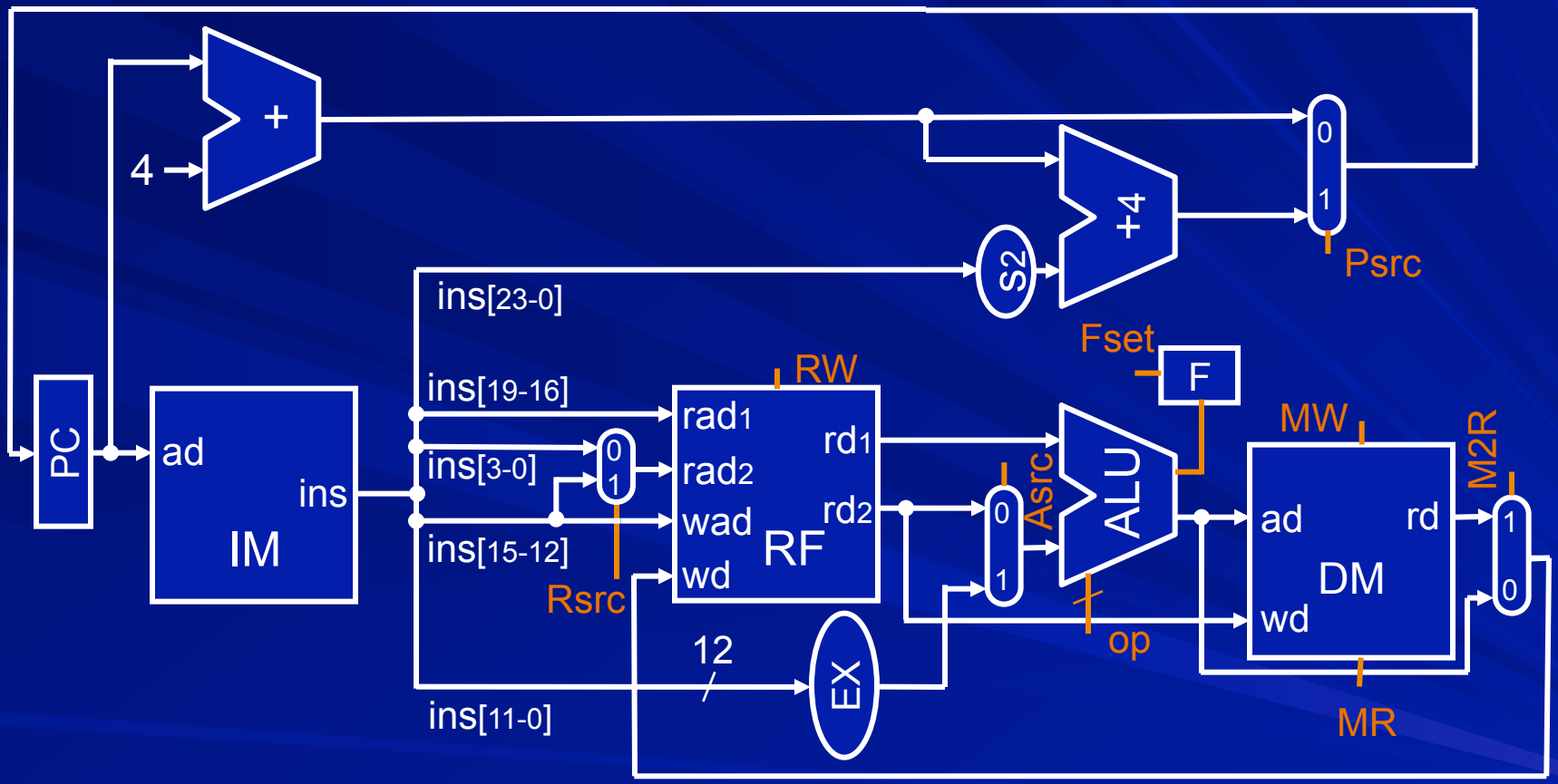
$$\max\left\{\begin{array}{c} t_{+} \\ t_I + t_R + t_A + t_R \end{array}\right.$$

Delay for {cmp, tst, …}



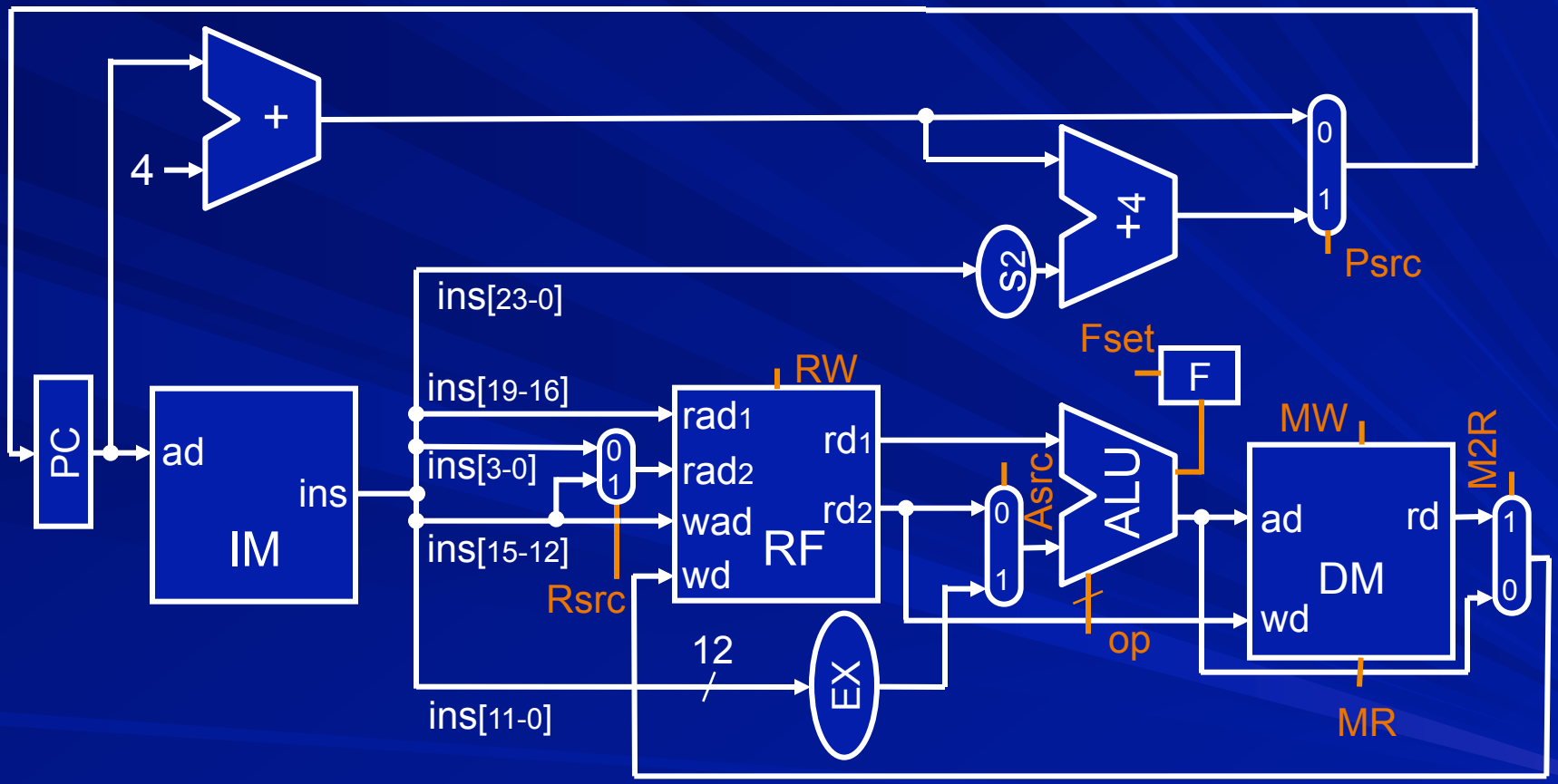
$$\max\begin{cases} t_{+} \\ t_I + t_R + t_A \end{cases}$$

Delay for {str}



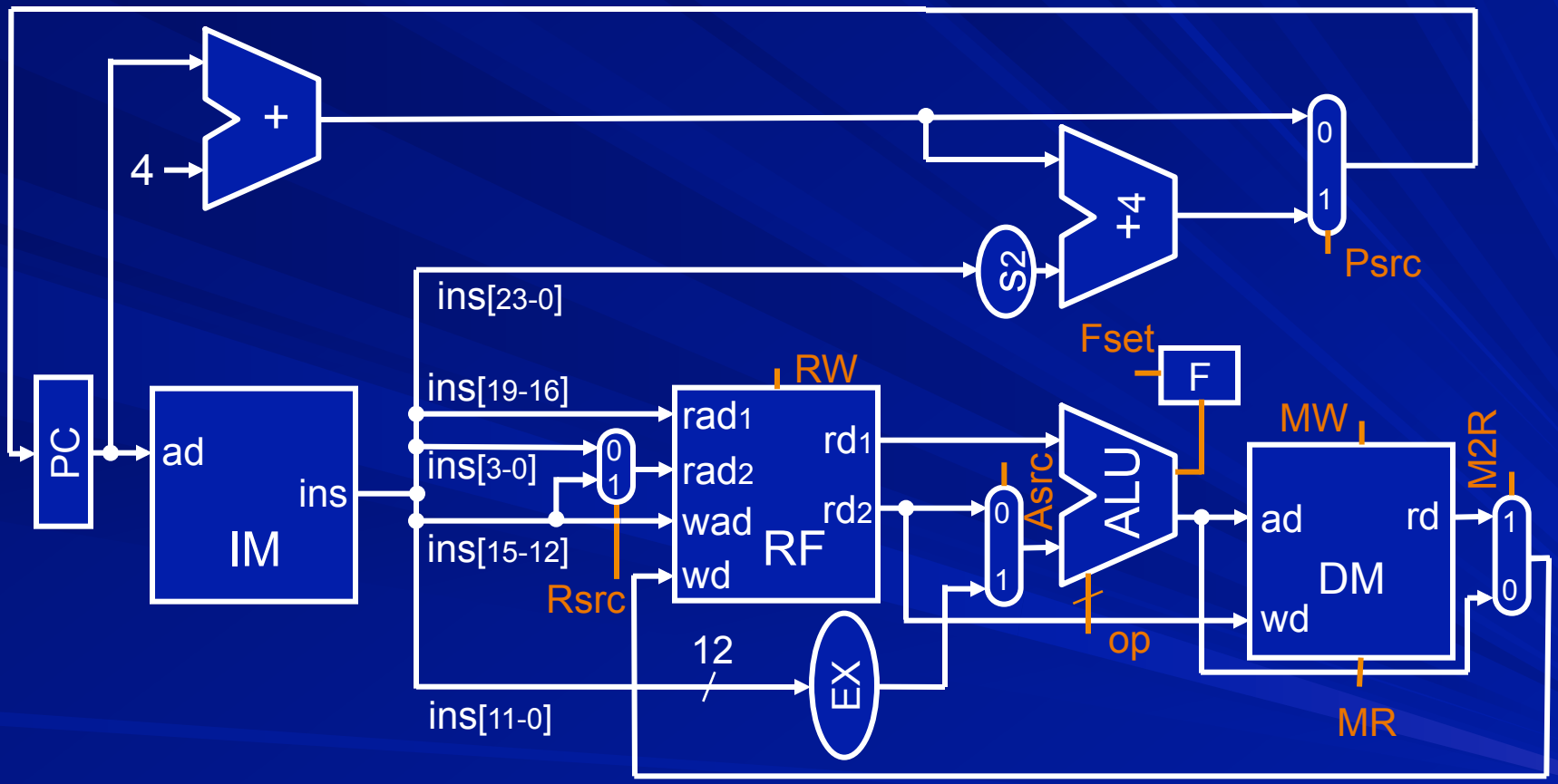
$$\max\begin{cases} t_{+} \\ t_I + t_R + t_A + t_M \end{cases}$$

Delay for {ldr}



$$\max\left\{\begin{array}{c} t_{+} \\ t_I + t_R + t_A + t_M + t_R \end{array}\right.$$

Delay for {b}



$$\max\begin{cases} t_I + t_+ \\ t_+ + t_+ \end{cases}$$

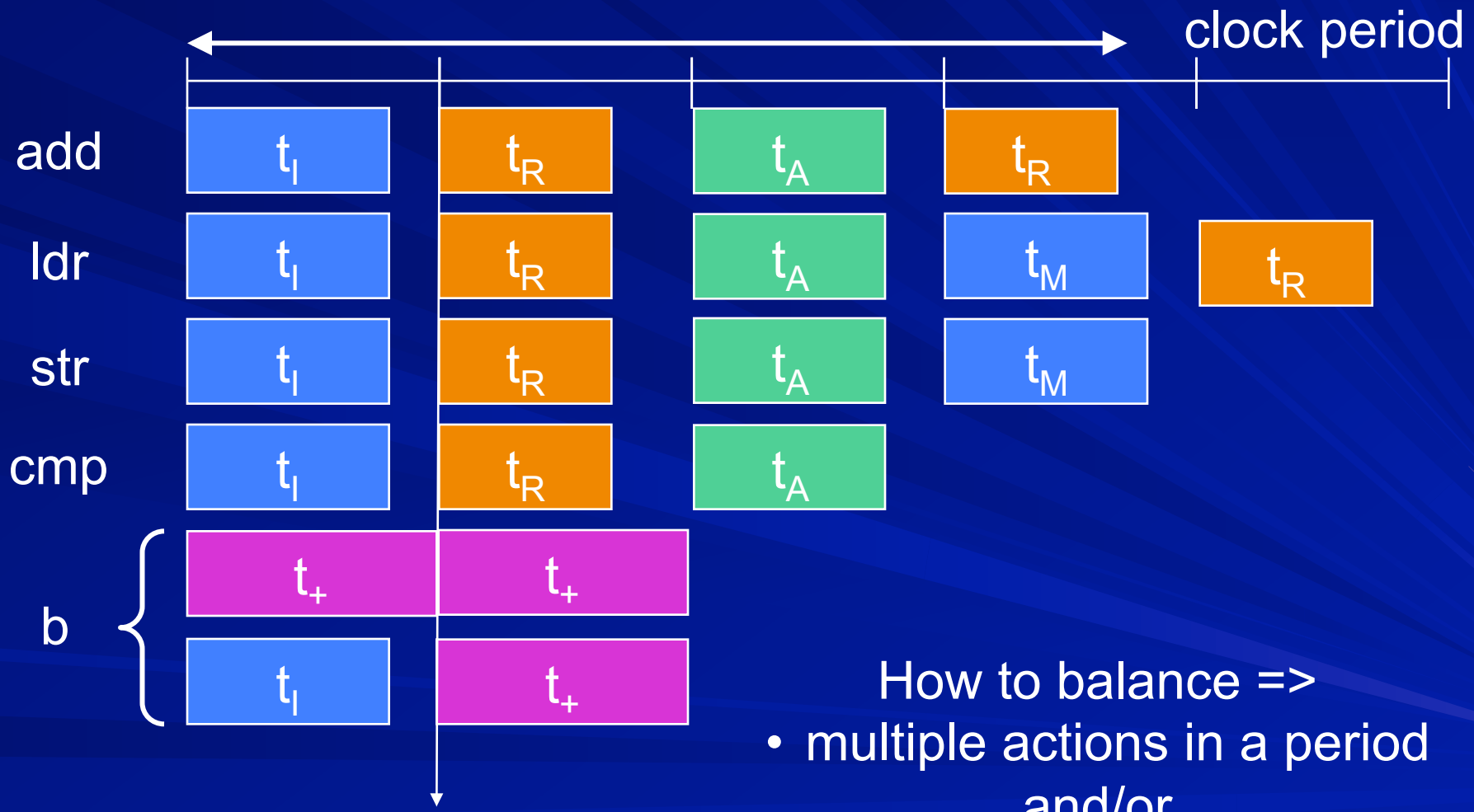
Overall clock period

$$\max \begin{cases} t_+, & t_I + t_R + t_A + t_R \\ t_+, & t_I + t_R + t_A \\ t_+, & t_I + t_R + t_A + t_M \\ t_+, & t_I + t_R + t_A + t_M + t_R \\ t_I + t_+, & t_+ + t_+ \end{cases} \quad ?$$

Clock period in single cycle design

Split the cycle into multiple cycles

Unbalanced delays



How to balance =>

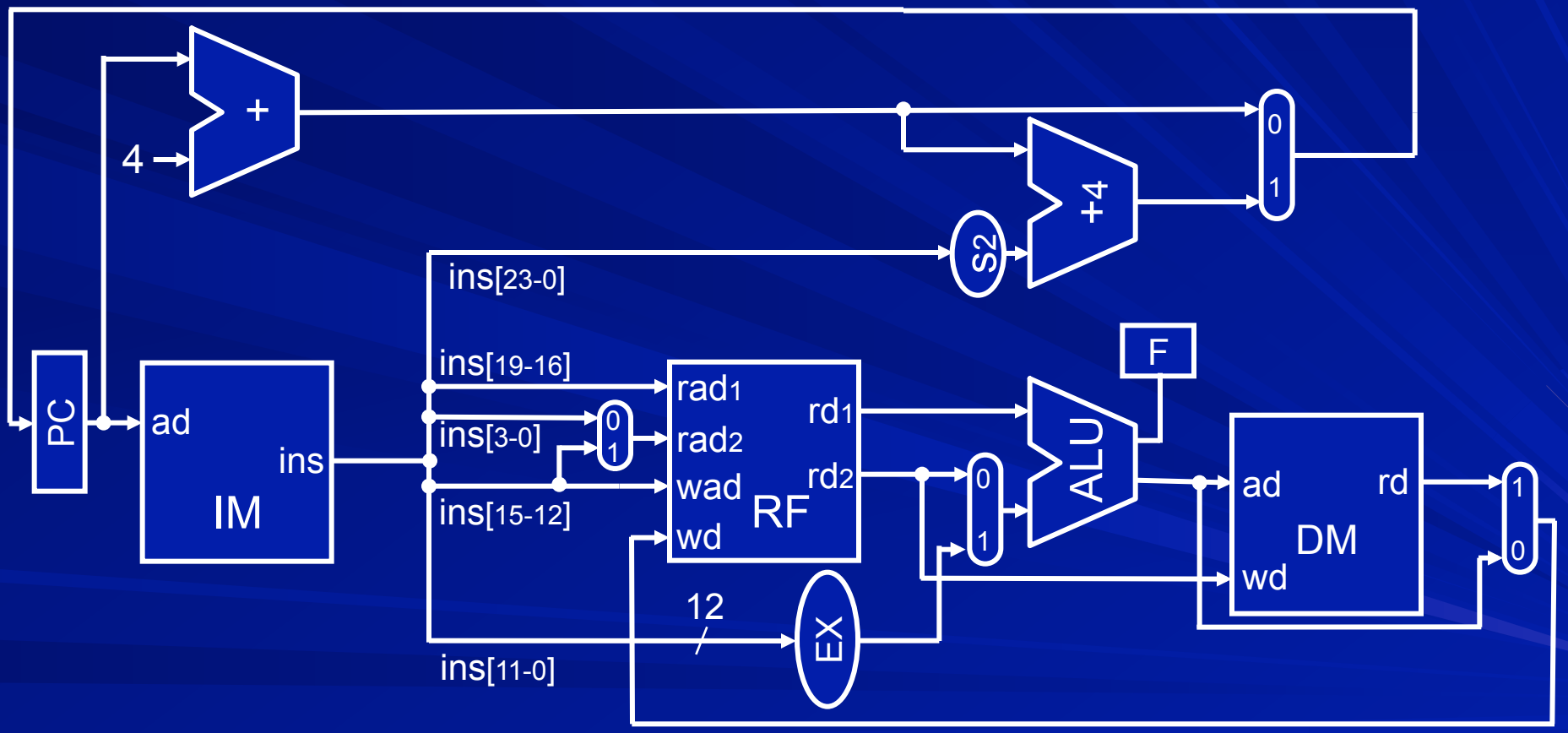
- multiple actions in a period
  and/or
- multiple periods for an action

Improving resource utilization

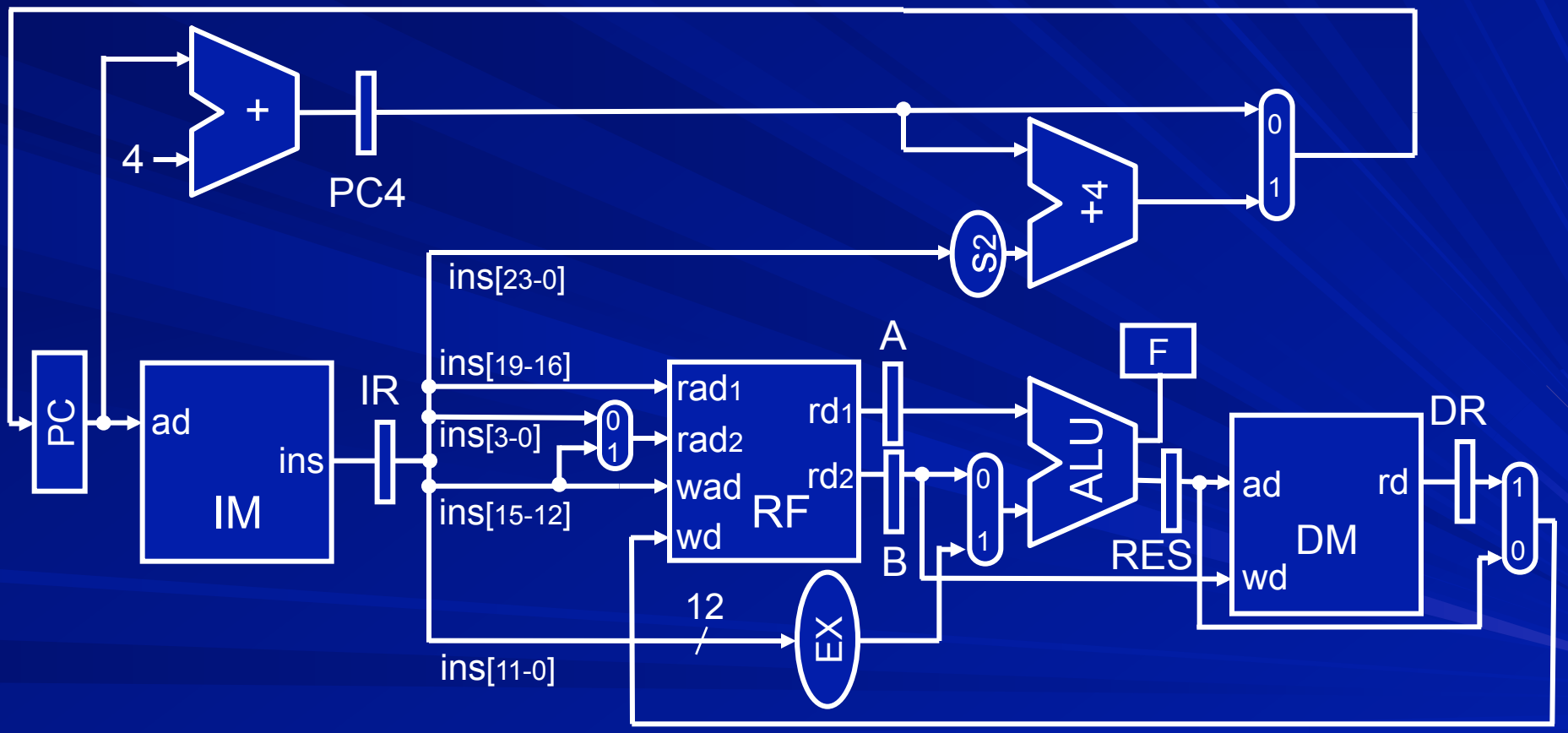
- Can we eliminate two adders?
- How to share (or reuse) a resource (say ALU) in different clock cycles?

- Store results in registers.
- Of course, more multiplexing may be required!
- Resources in this design: RF, ALU, MEM.

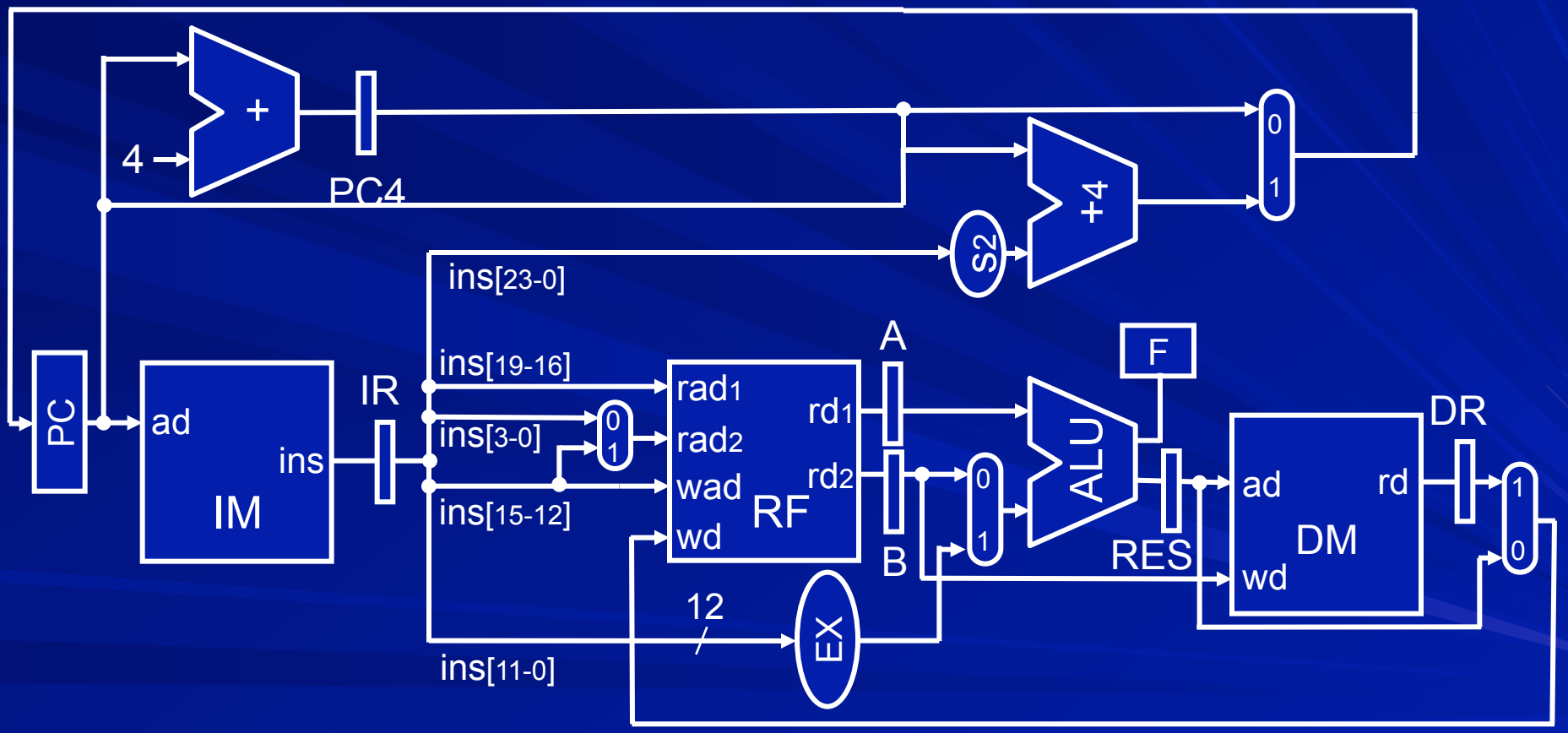
Single Cycle Datapath



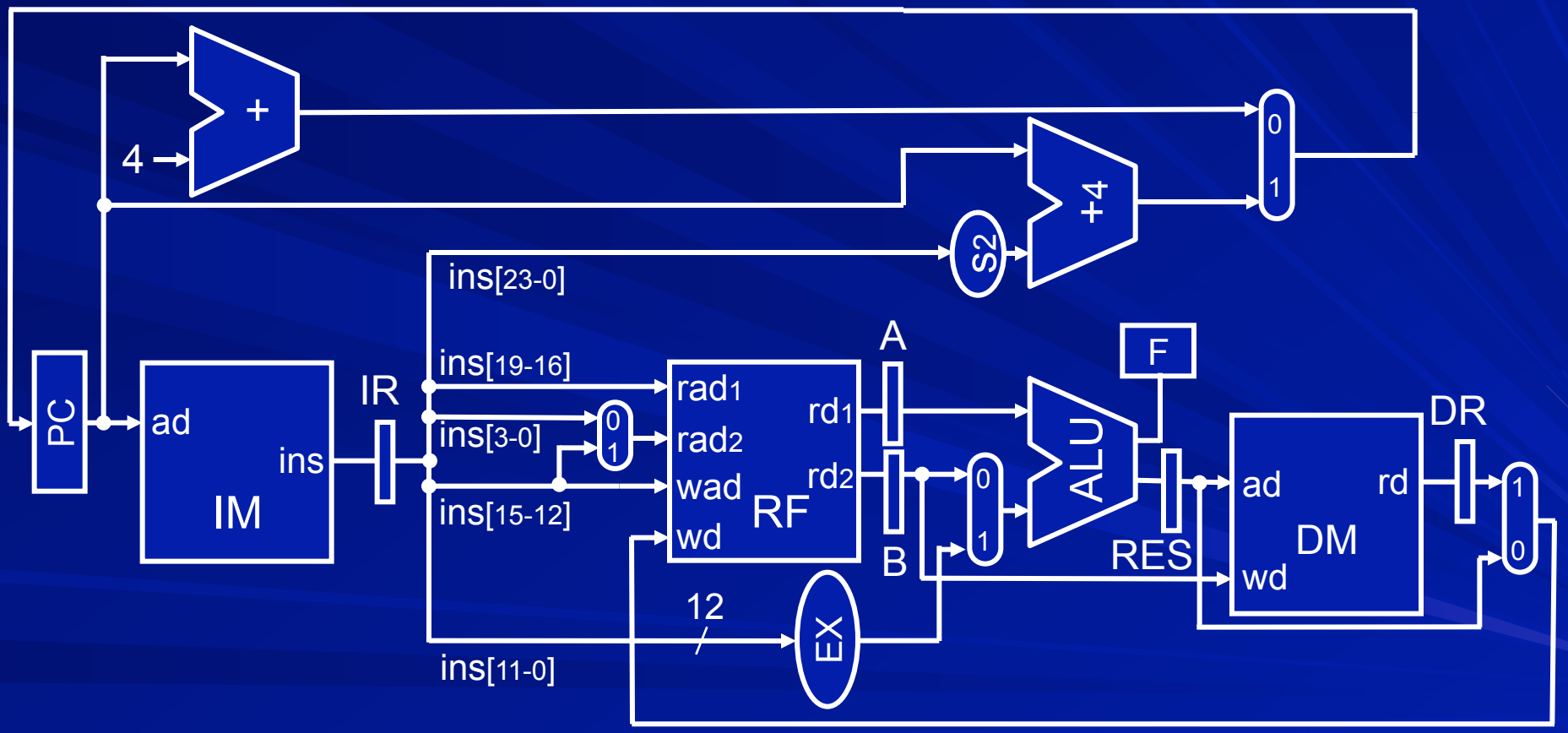
Introducing registers



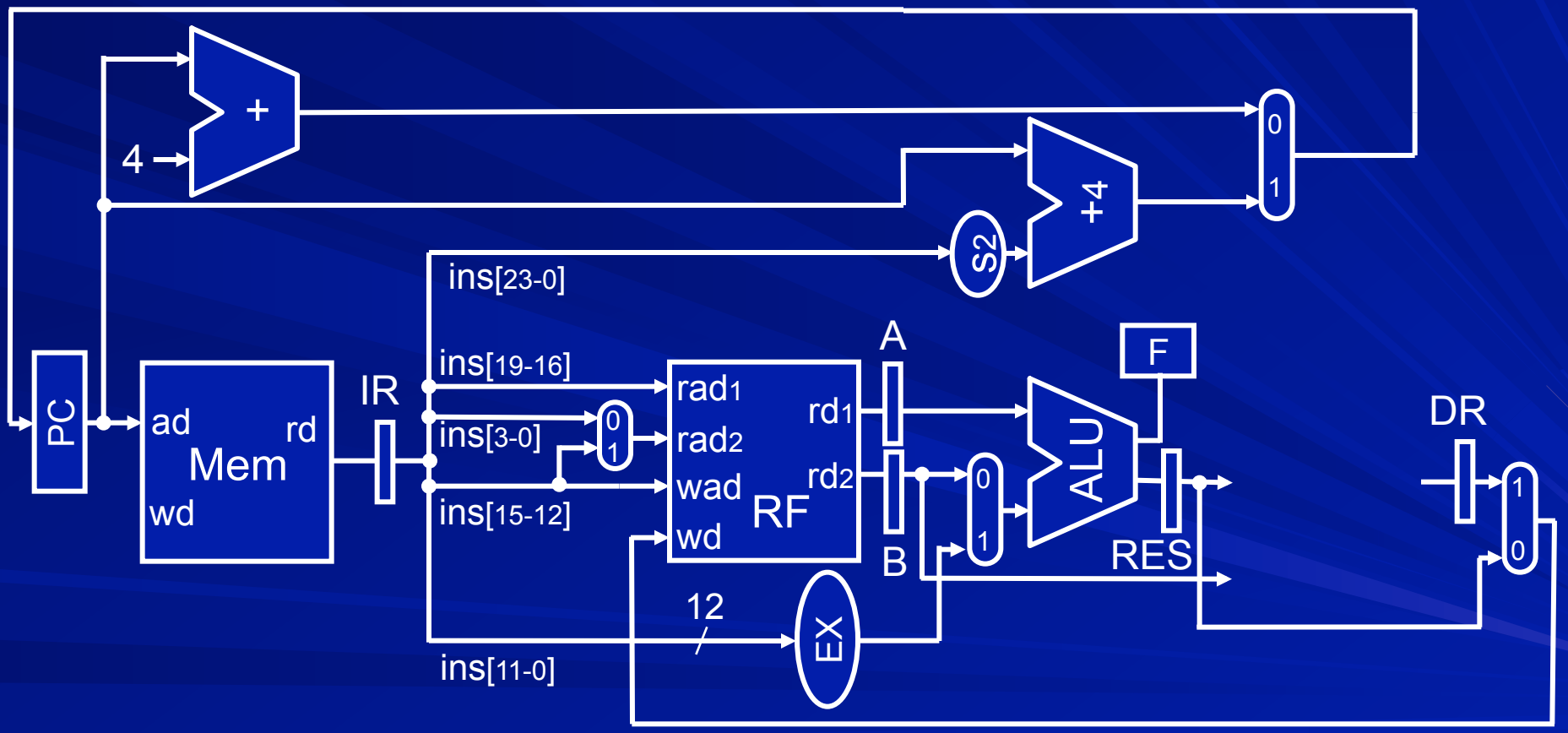
PC4 can be eliminated



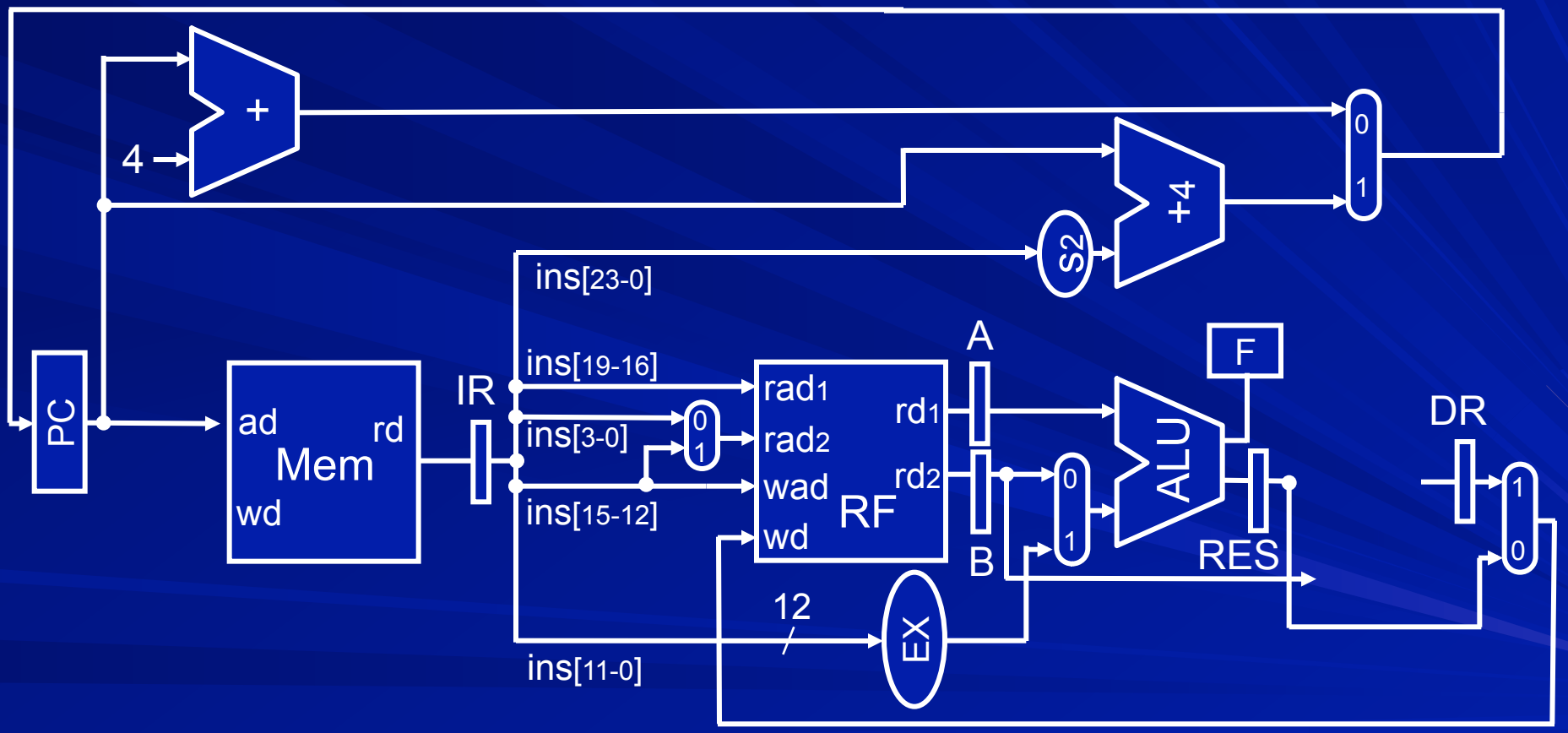
Merge IM and DM



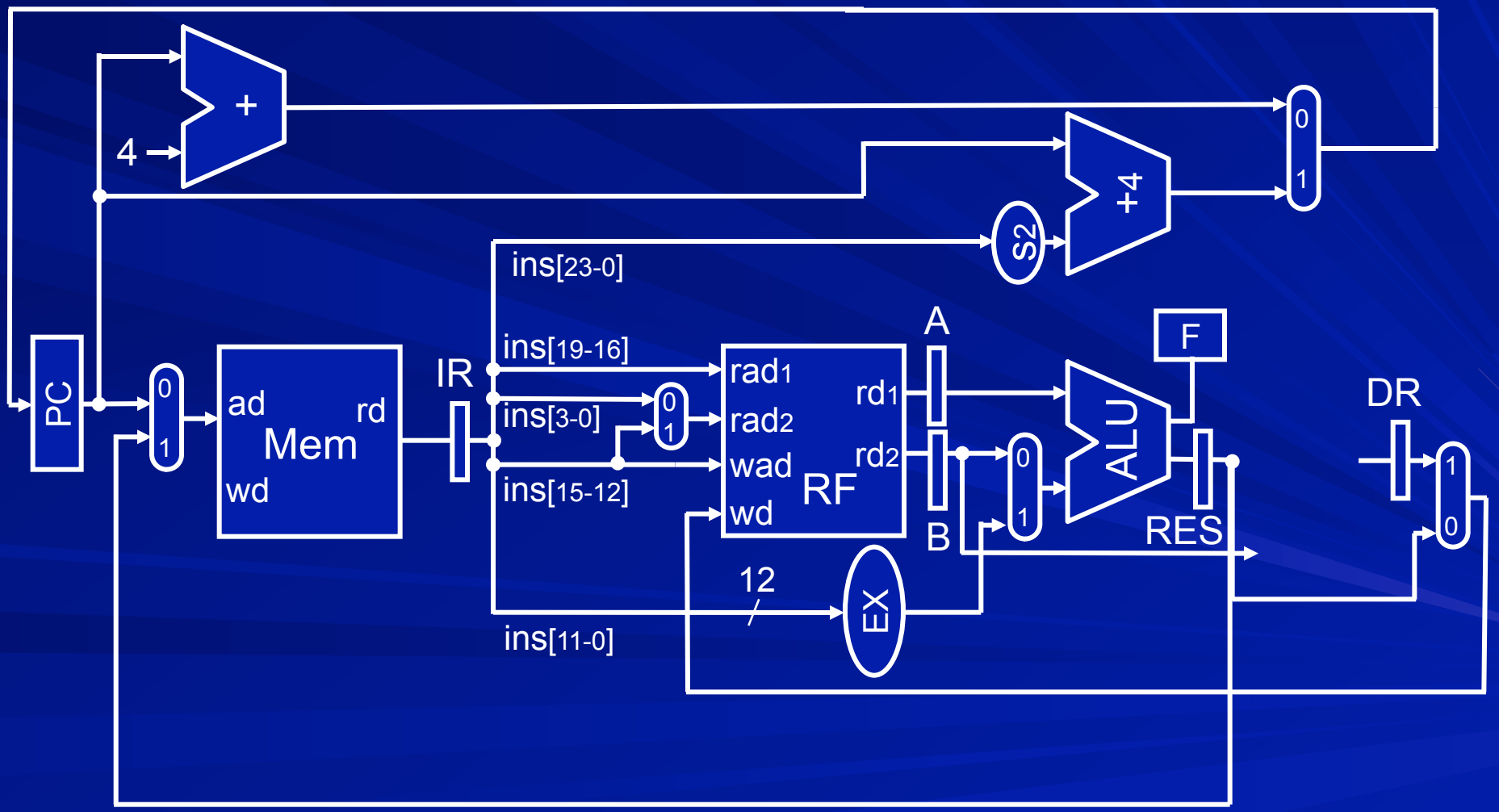
Merge IM and DM



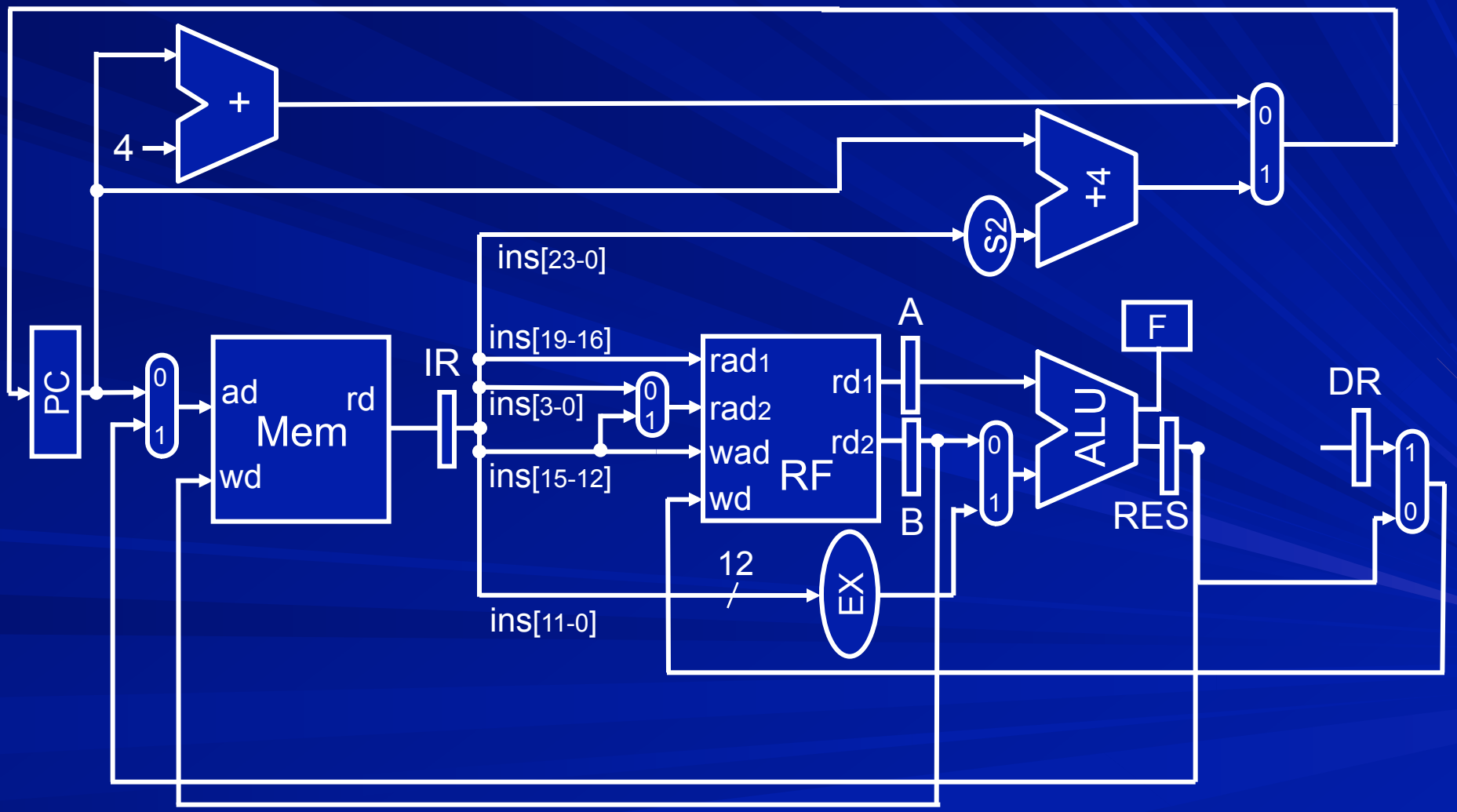
Merge IM and DM



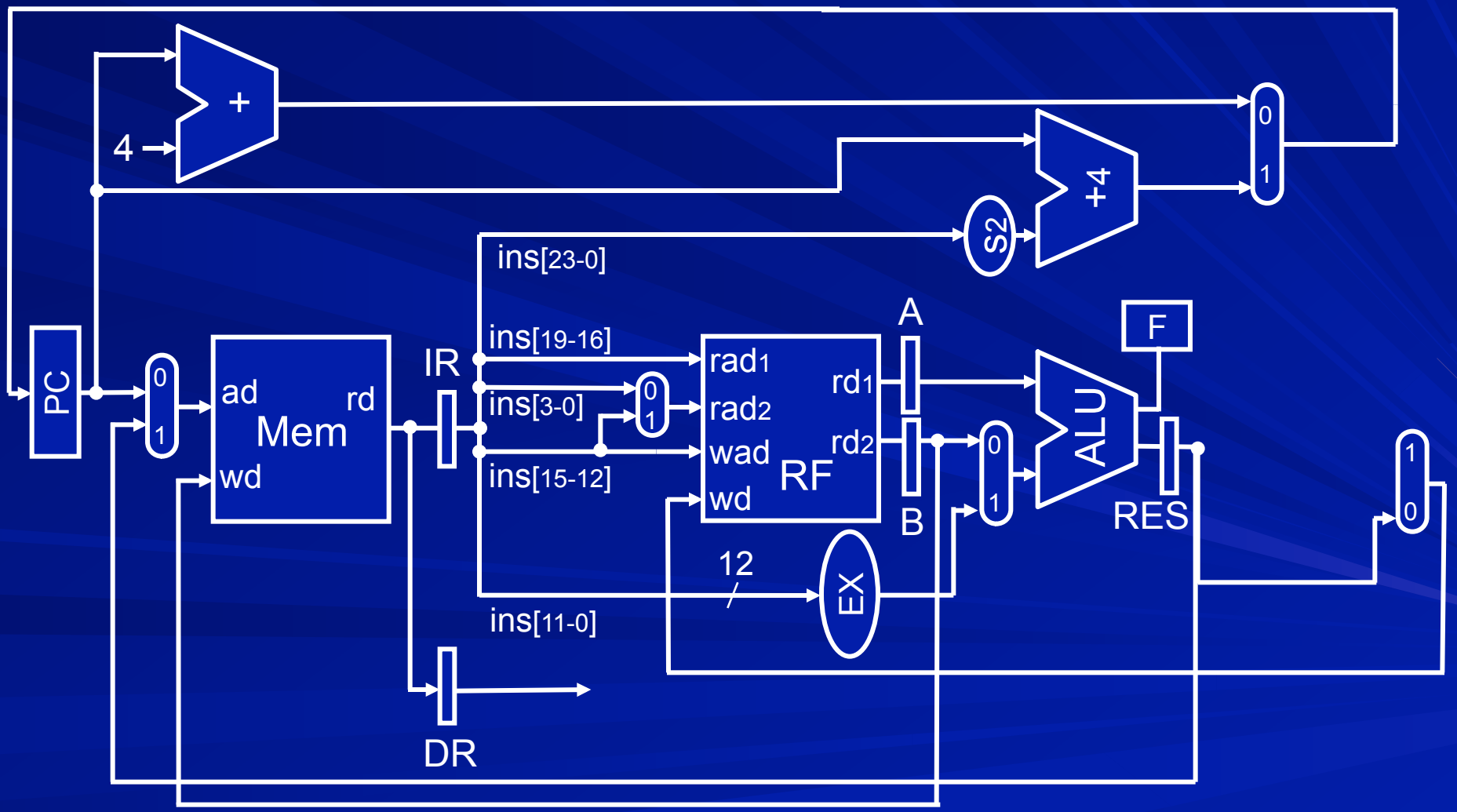
Merge IM and DM



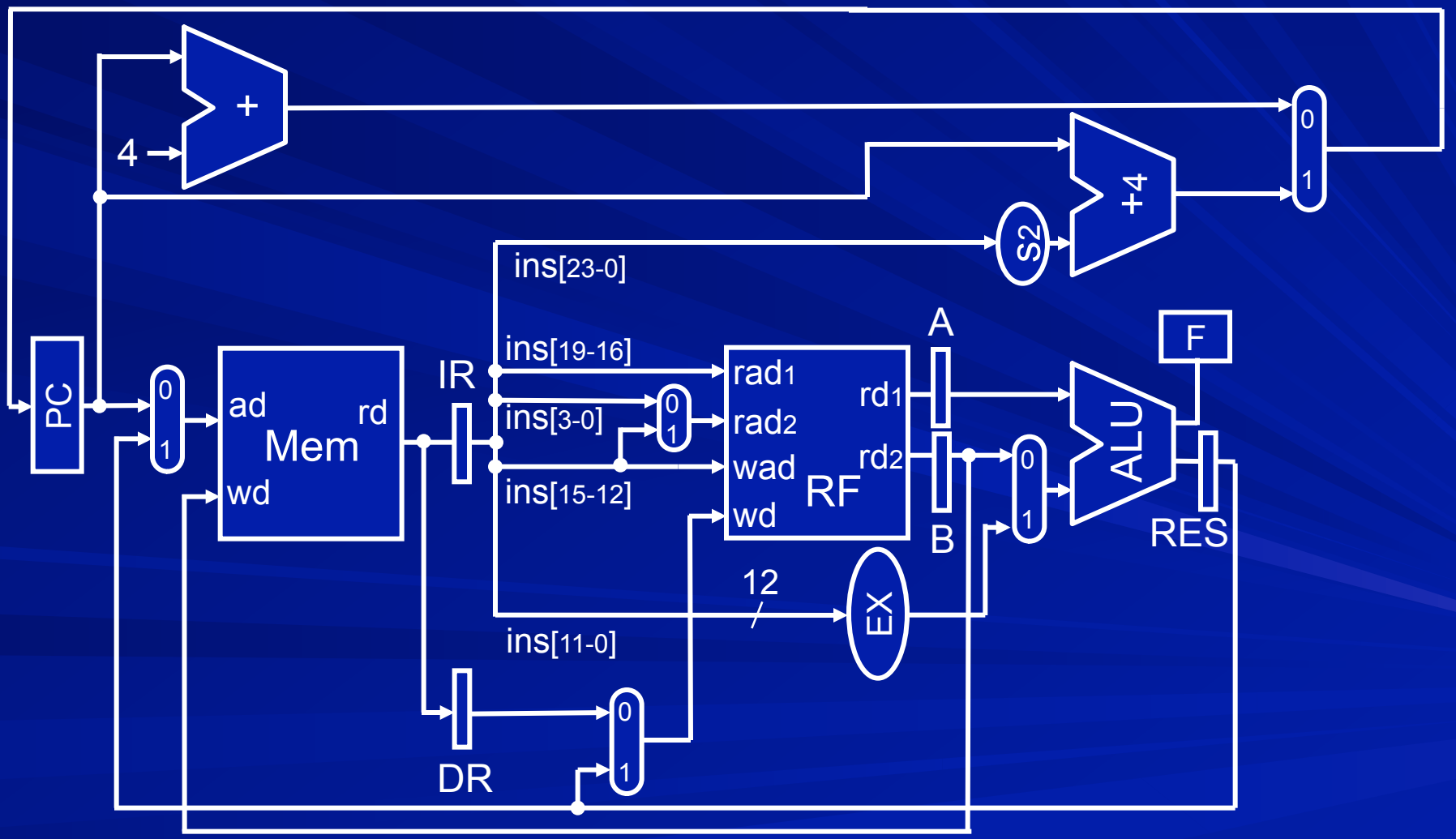
Merge IM and DM



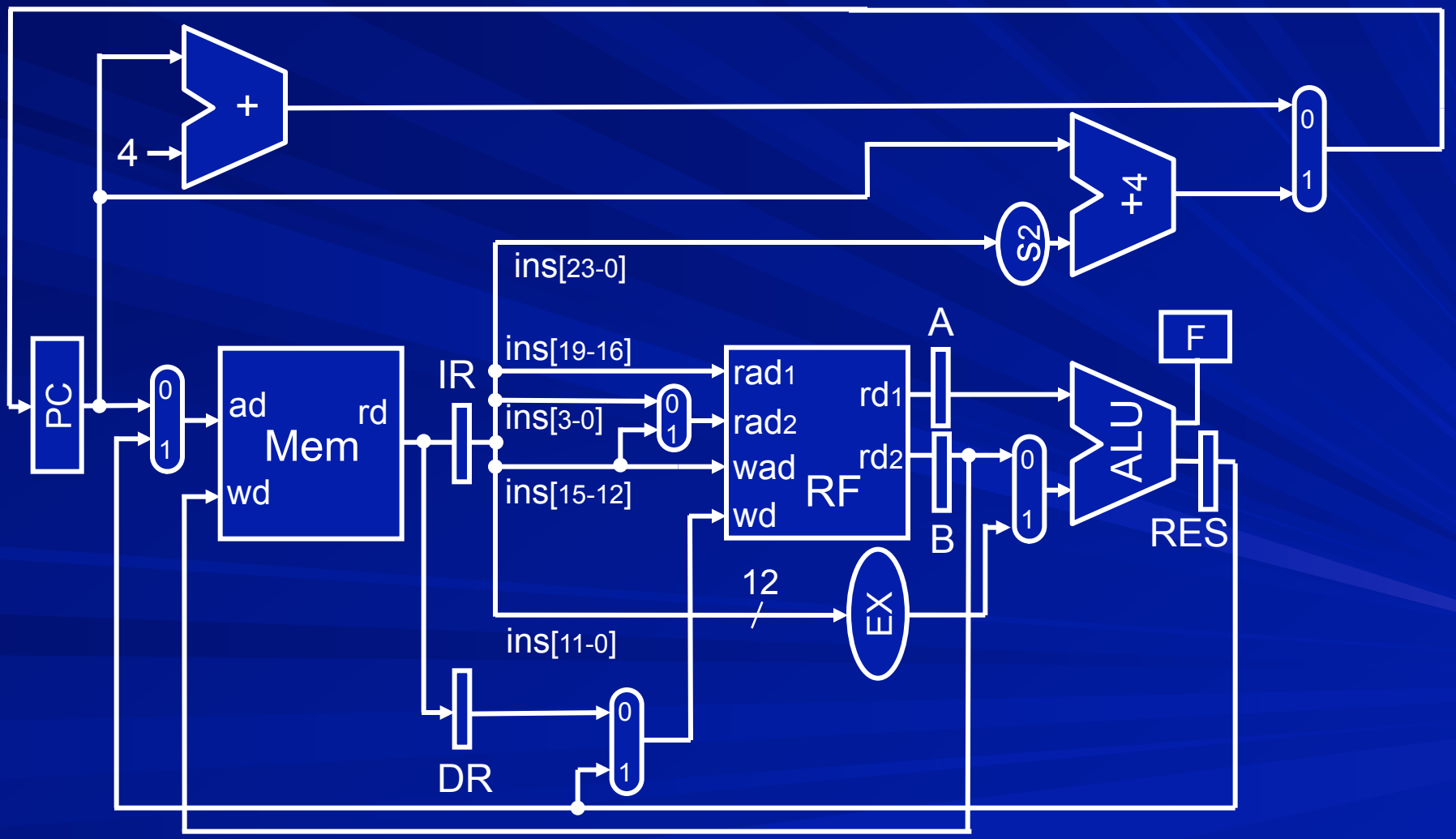
Merge IM and DM



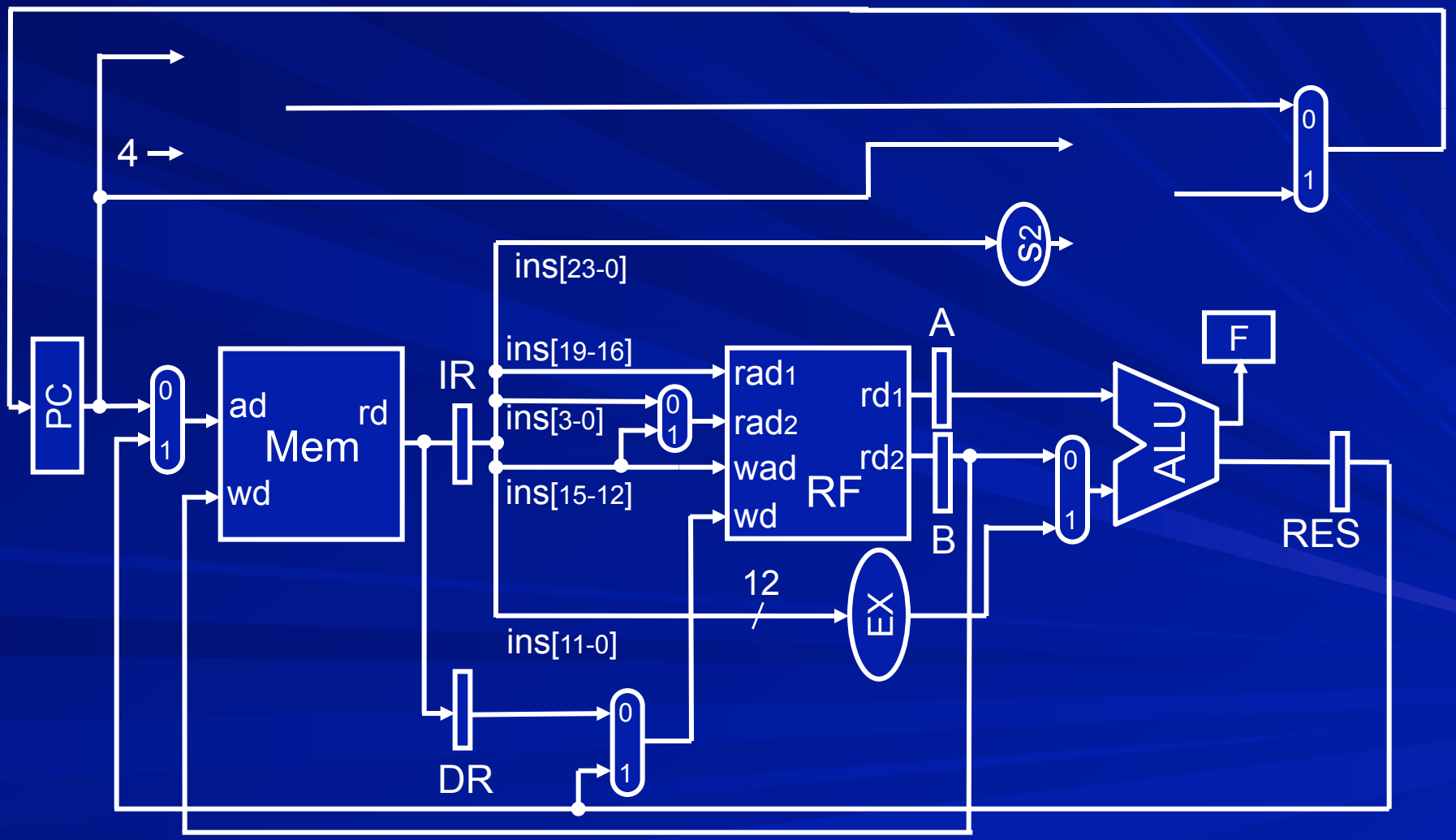
Merge IM and DM



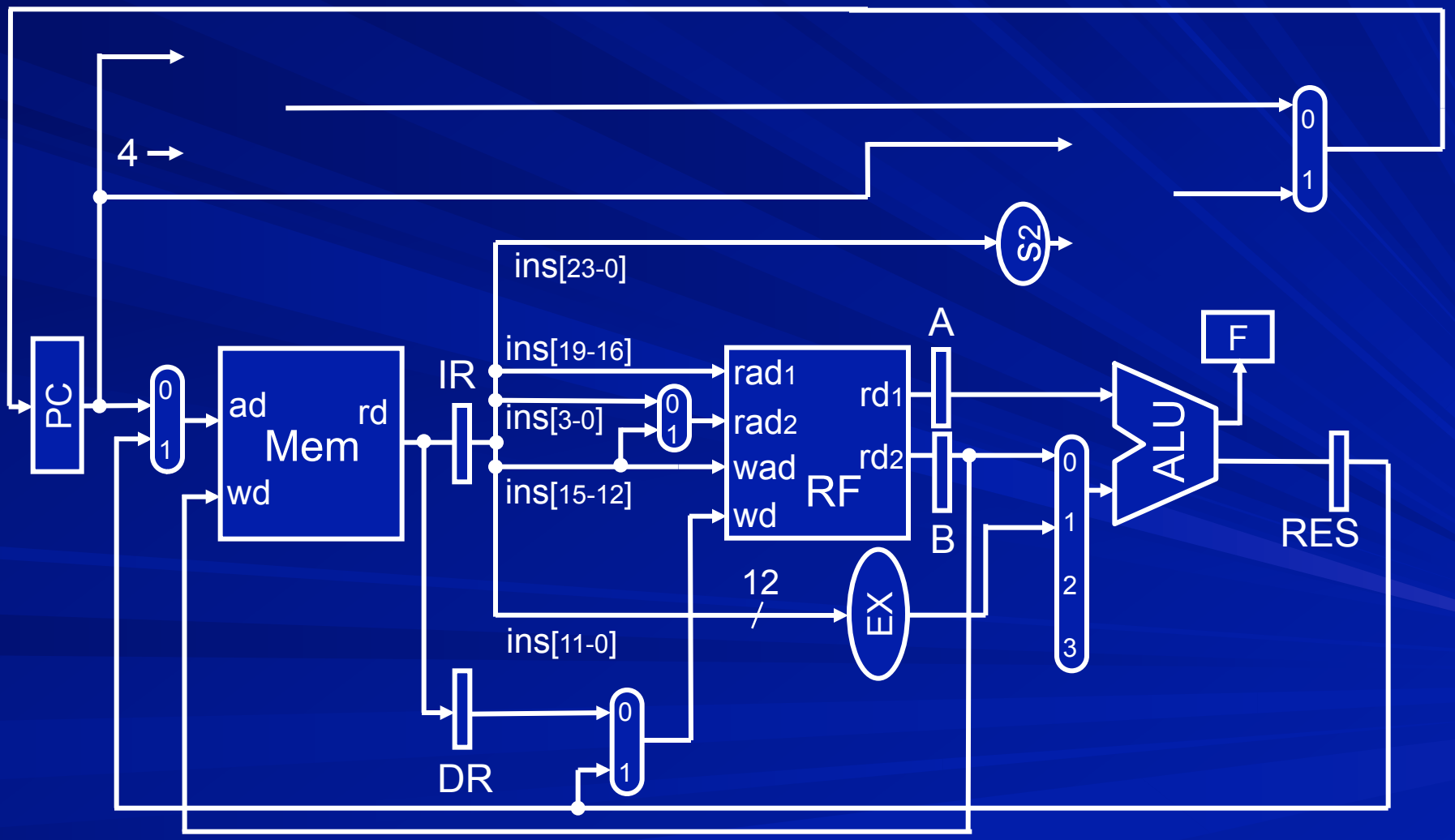
Eliminate adders



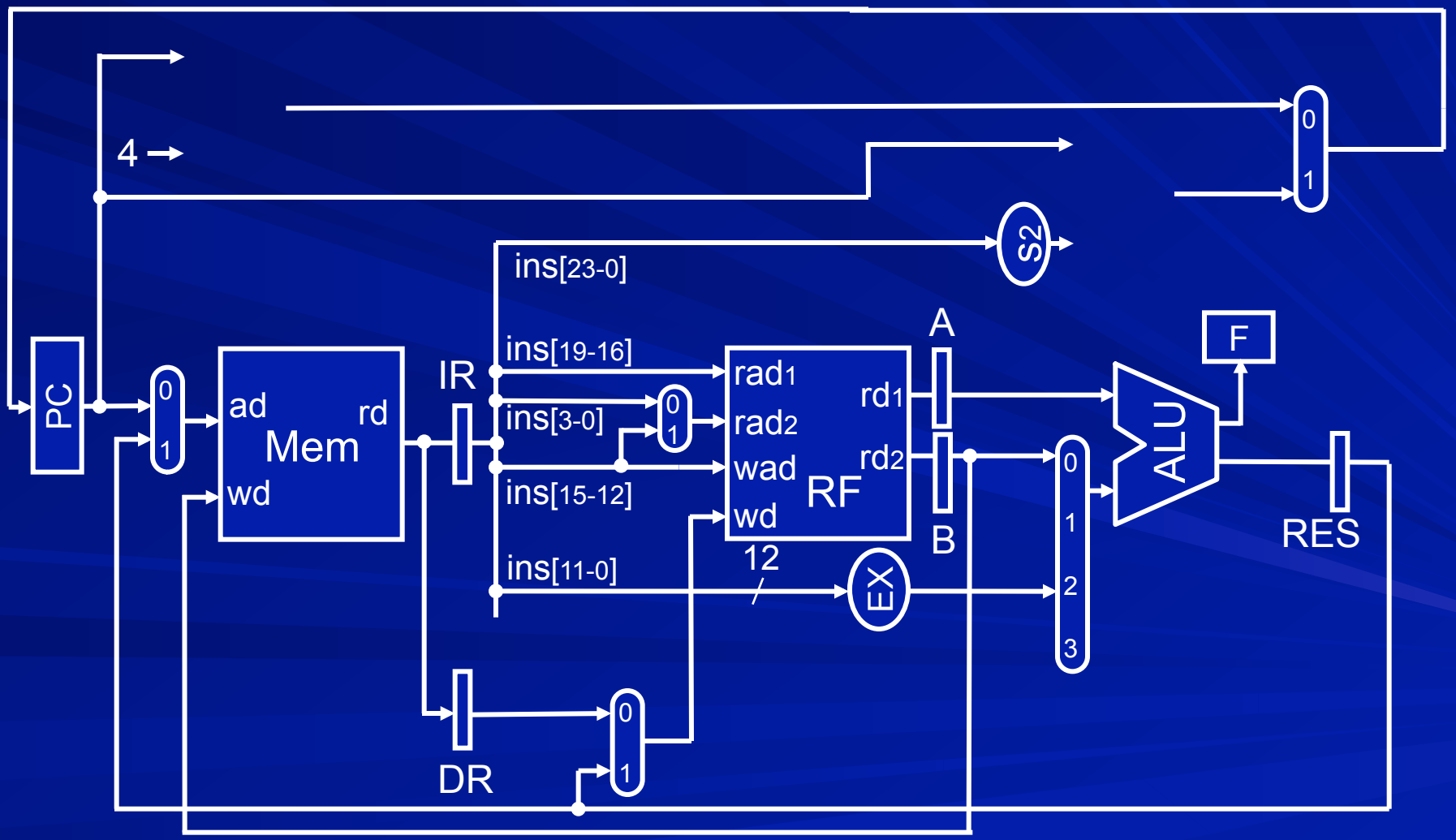
Eliminate adders



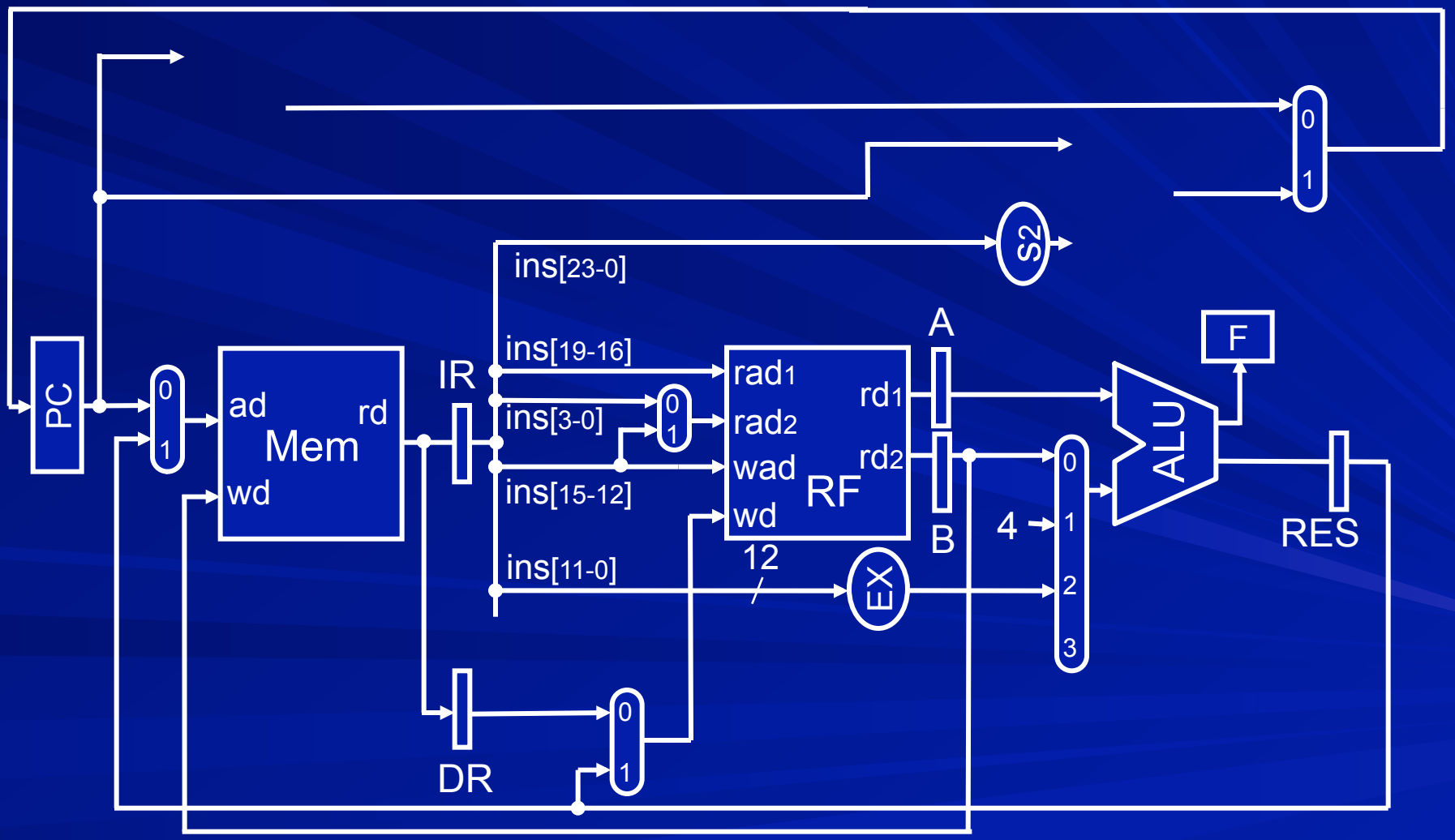
Eliminate adders



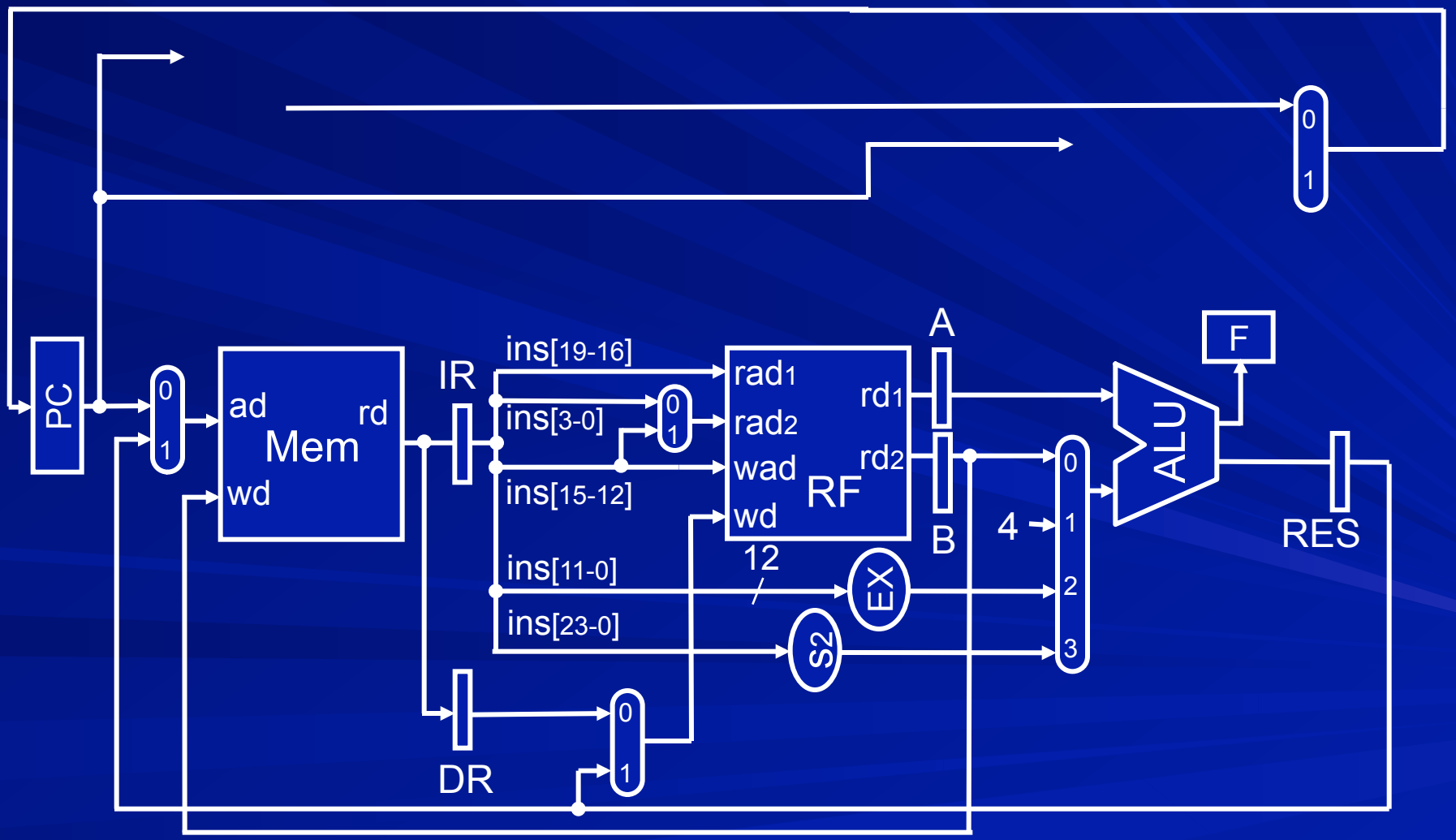
Eliminate adders



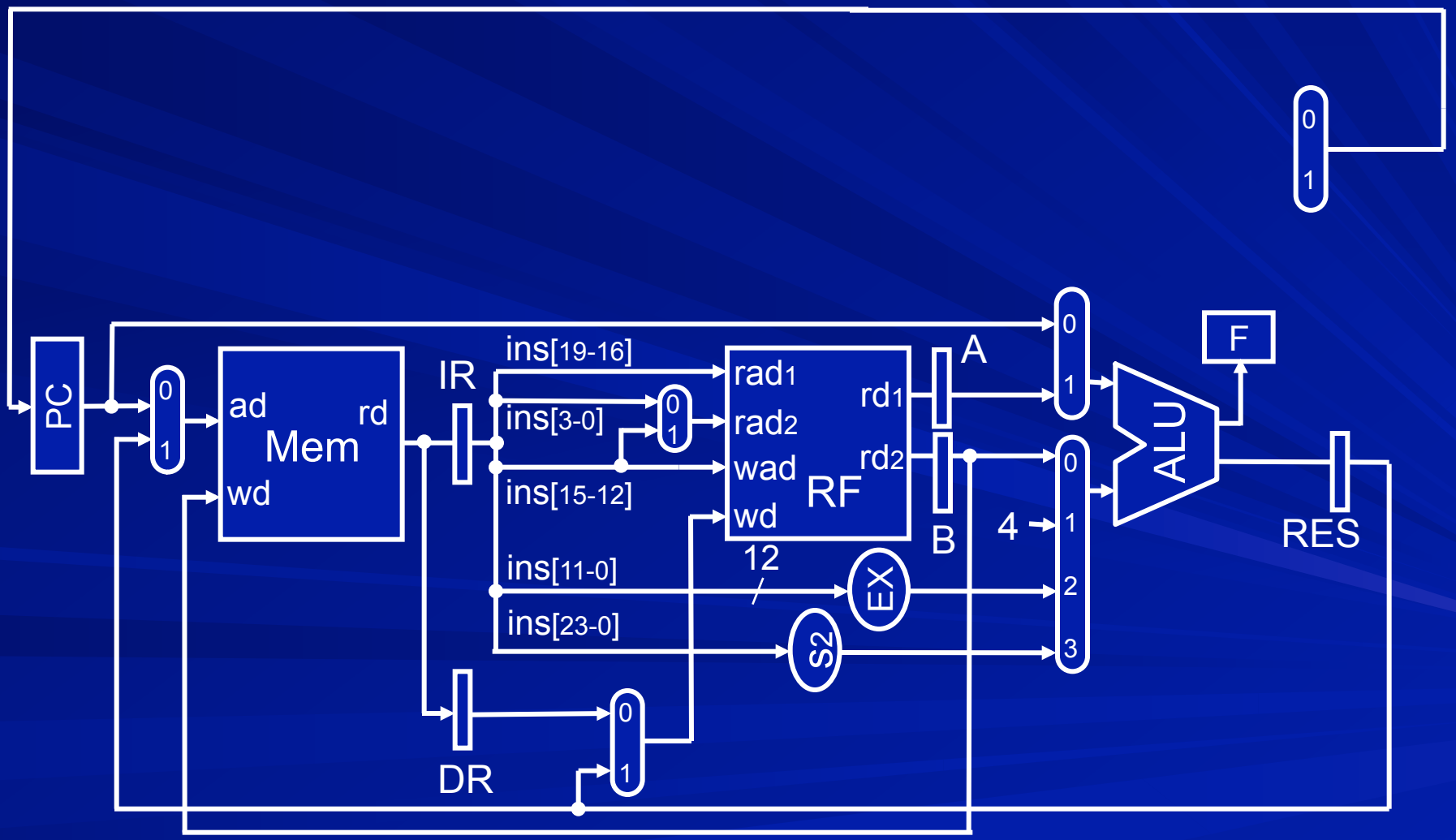
Eliminate adders



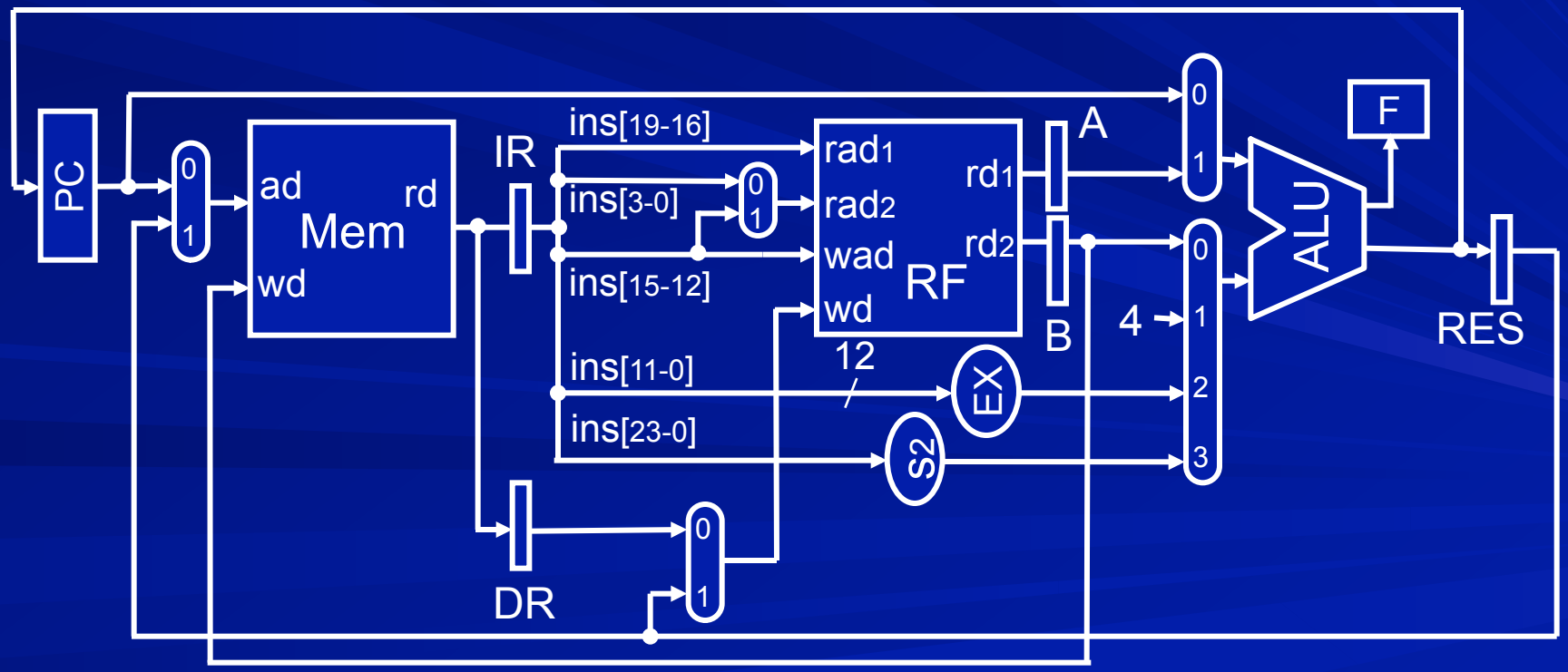
Eliminate adders



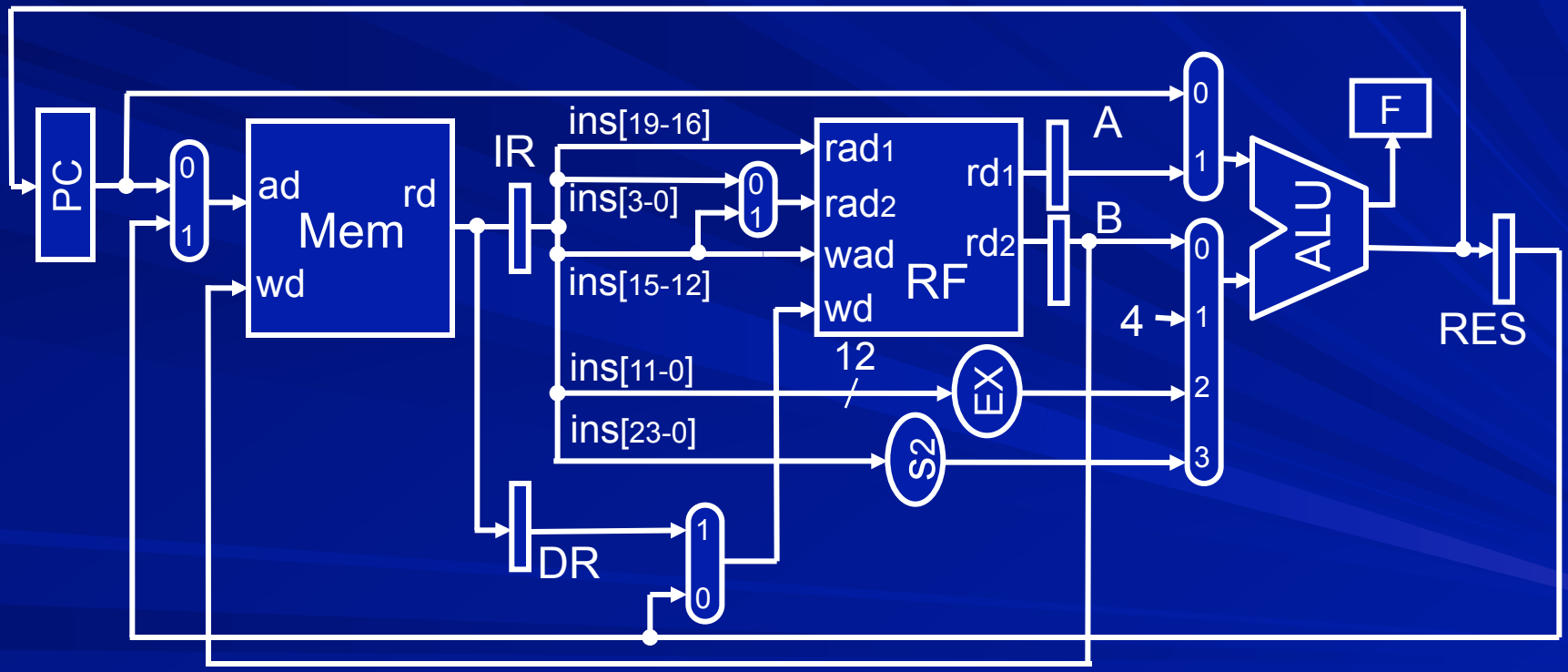
Eliminate adders



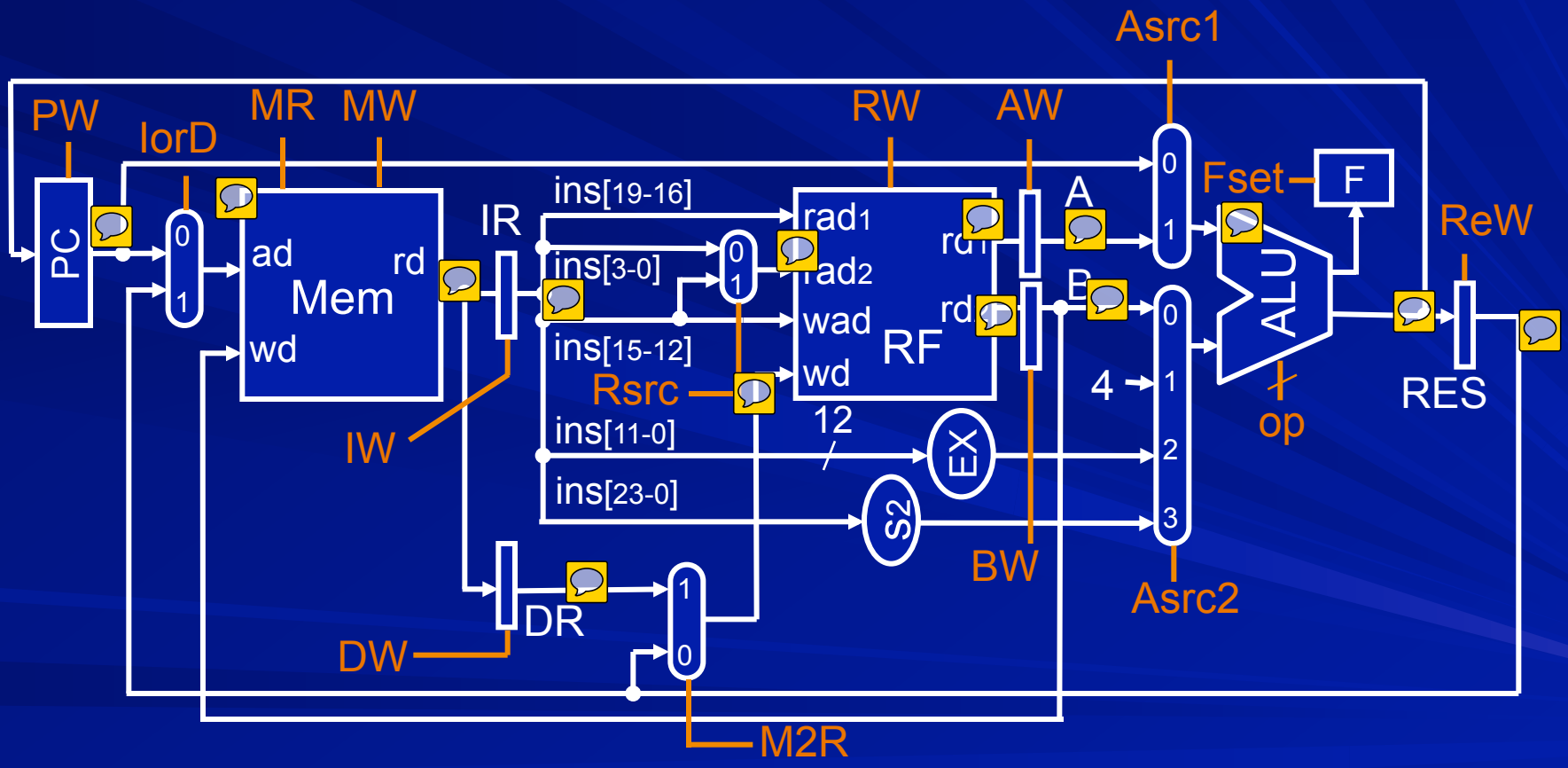
Eliminate adders



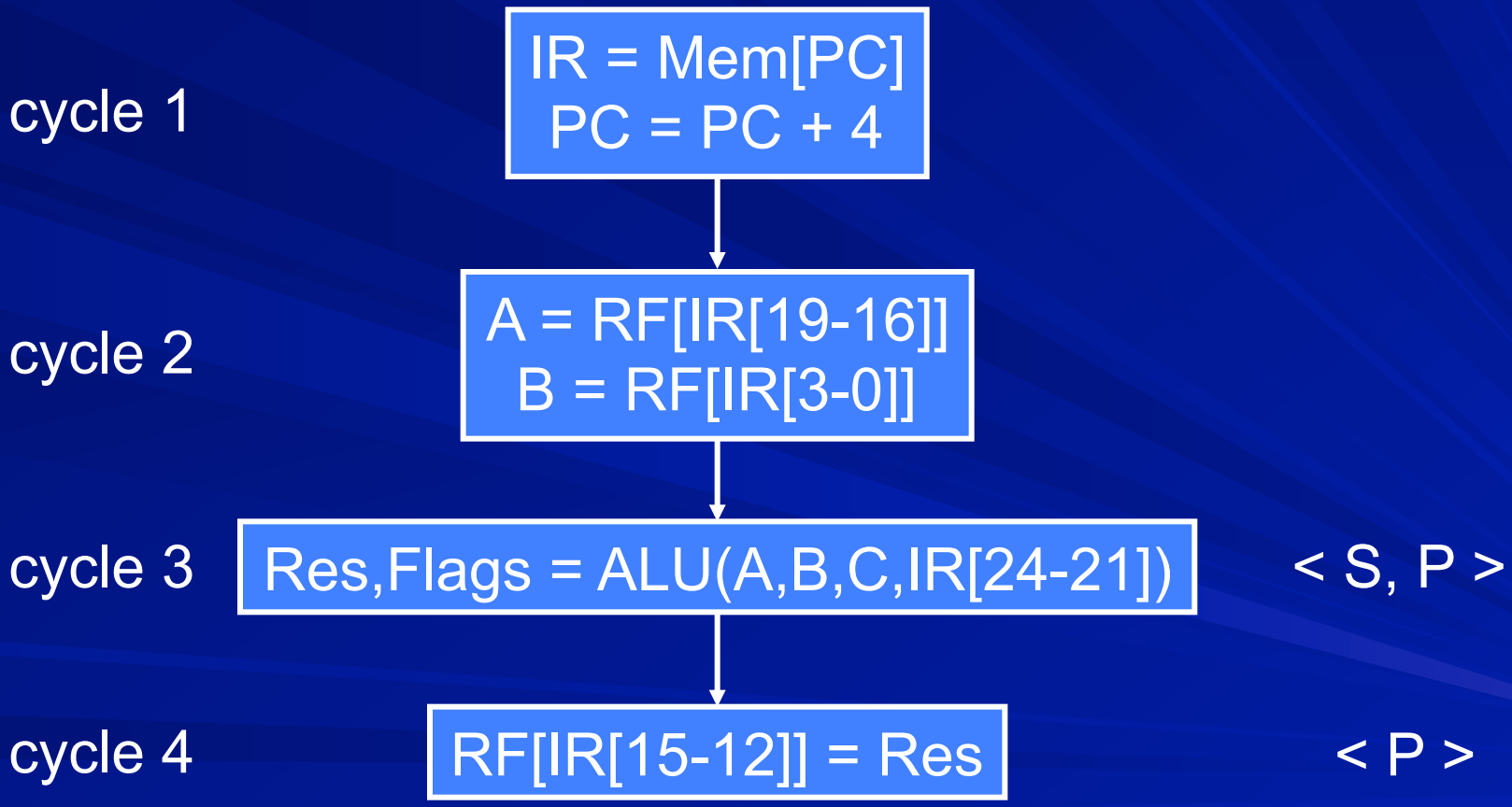
Multi-cycle Datapath



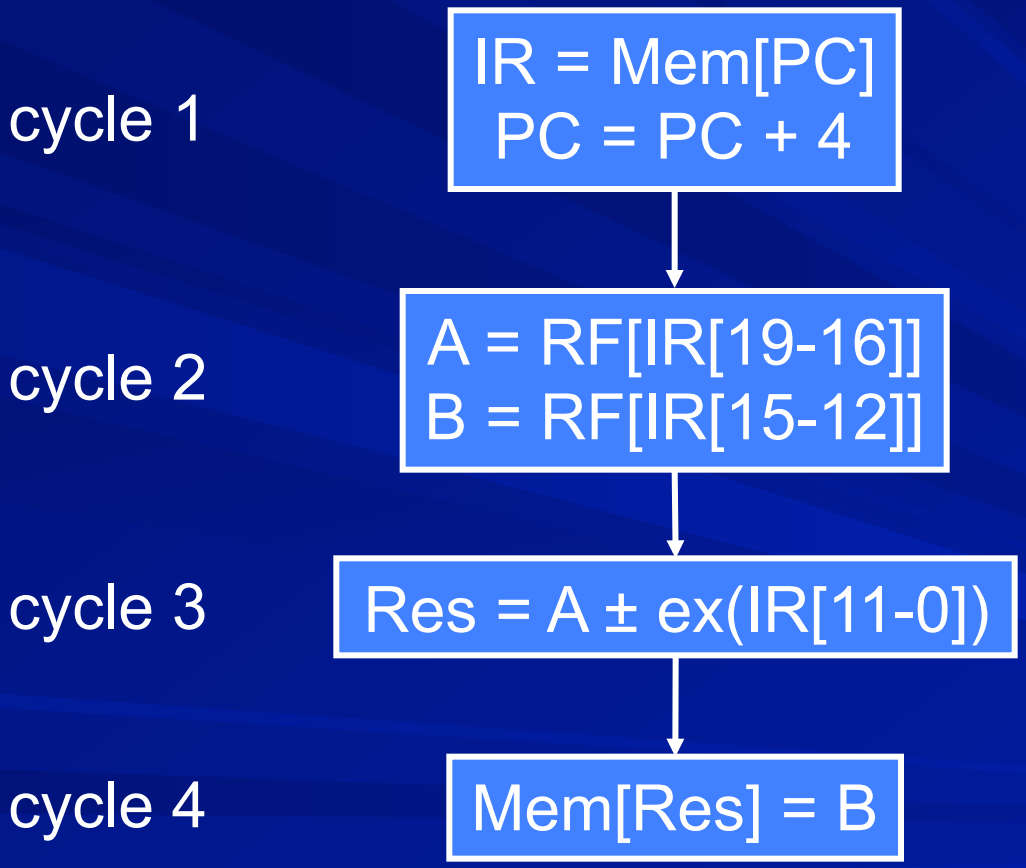
Control signals in multi-cycle datapath



Break Instruction Execution into Cycles: DP instructions

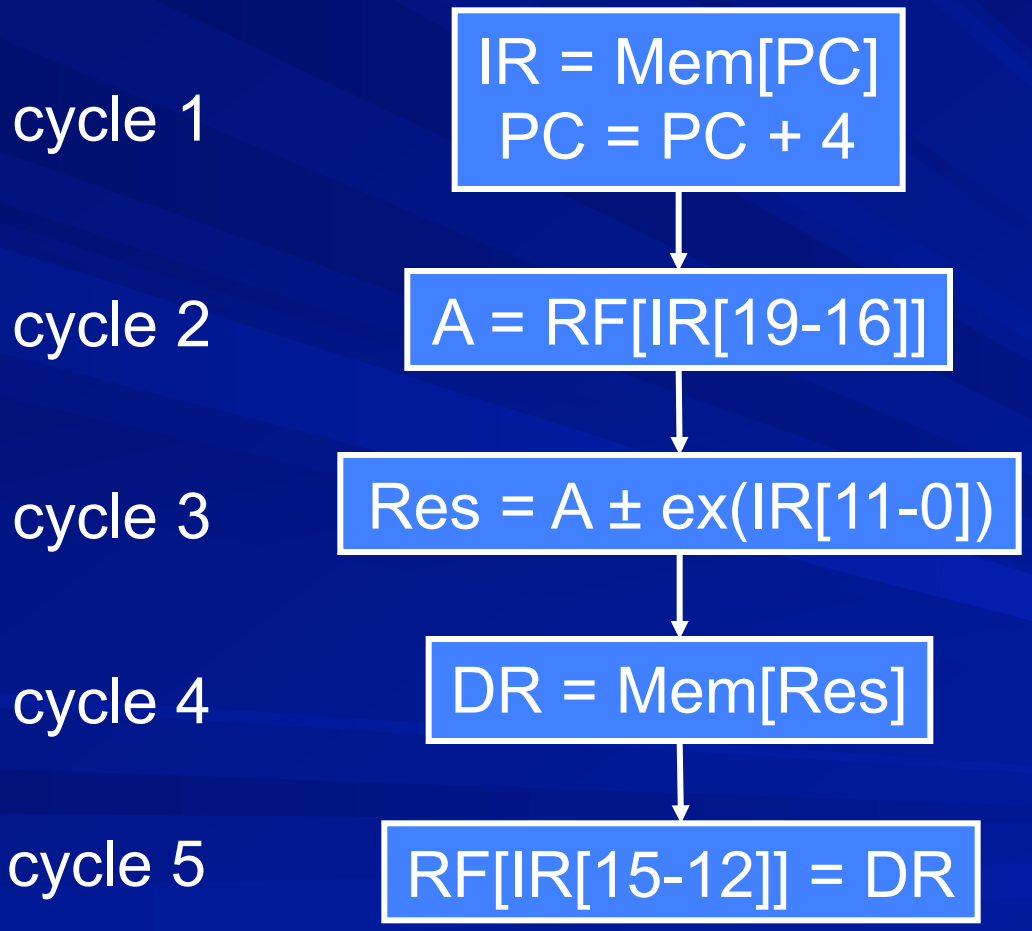


Break Instruction Execution into Cycles: str instruction



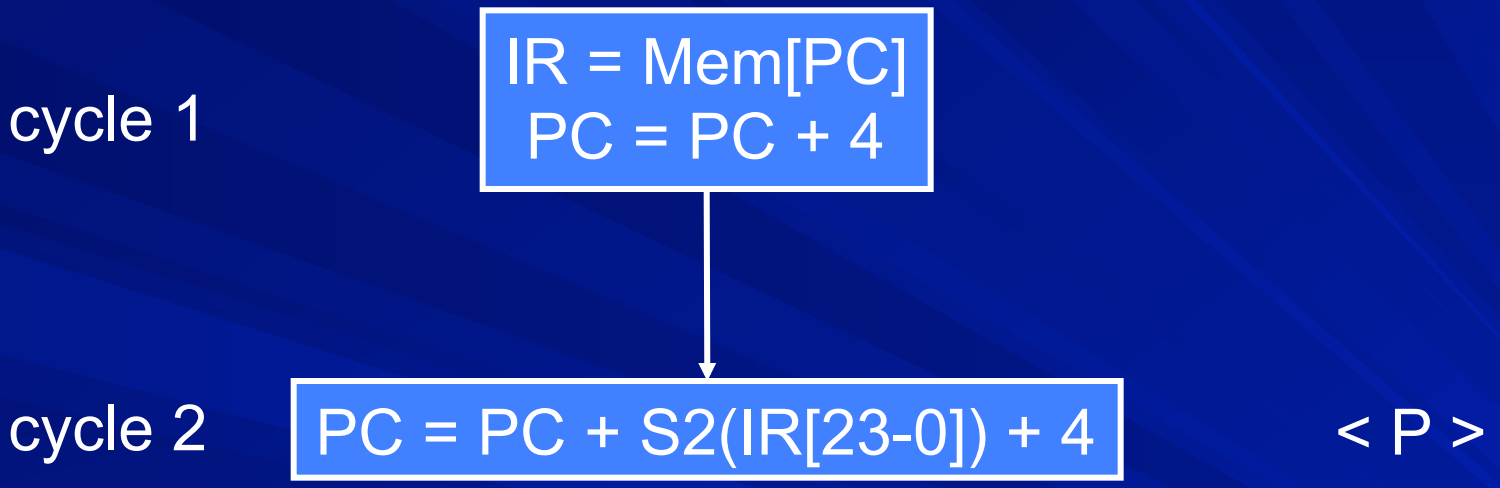
< P >

Break Instruction Execution into Cycles: ldr instruction

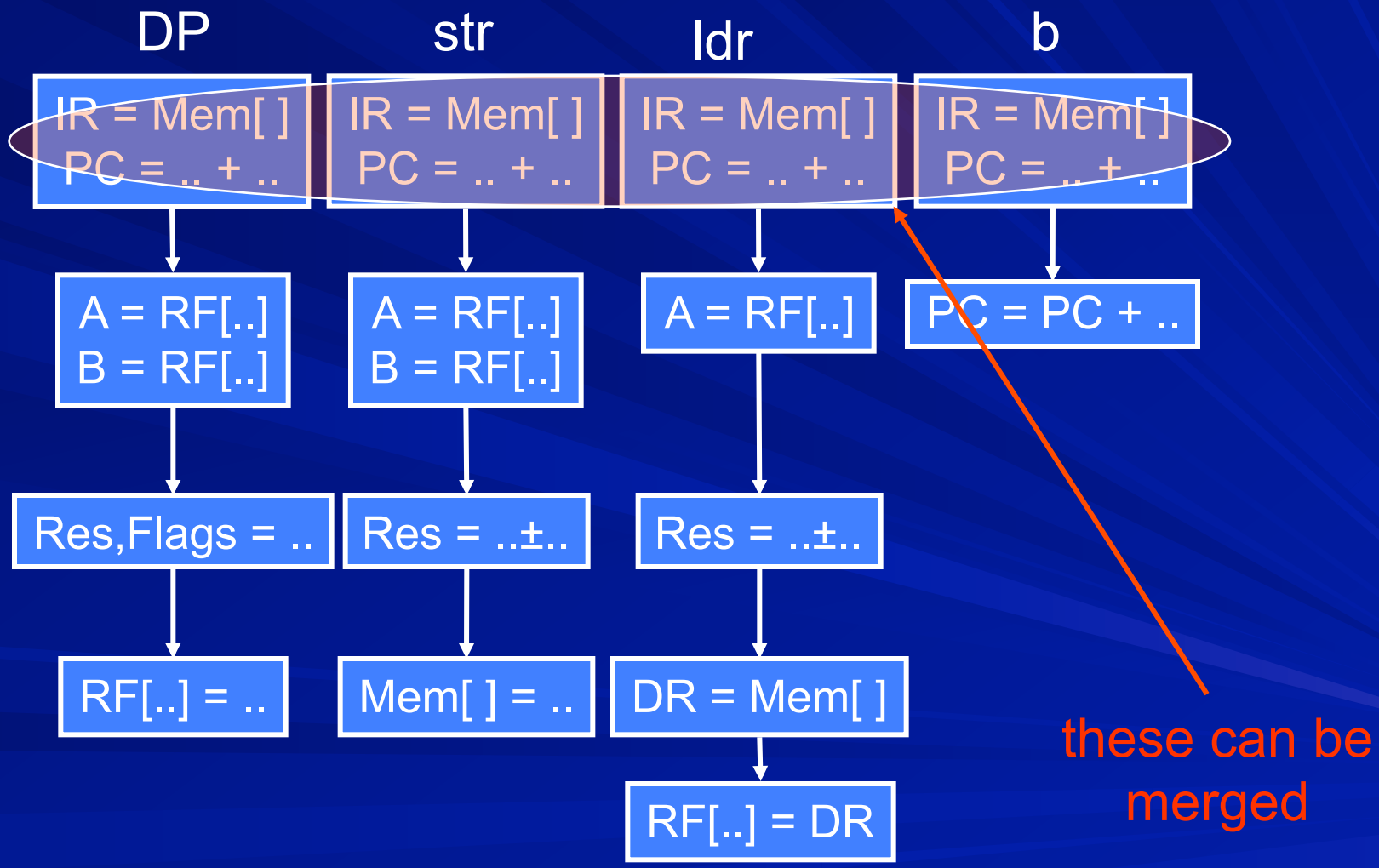


< P >

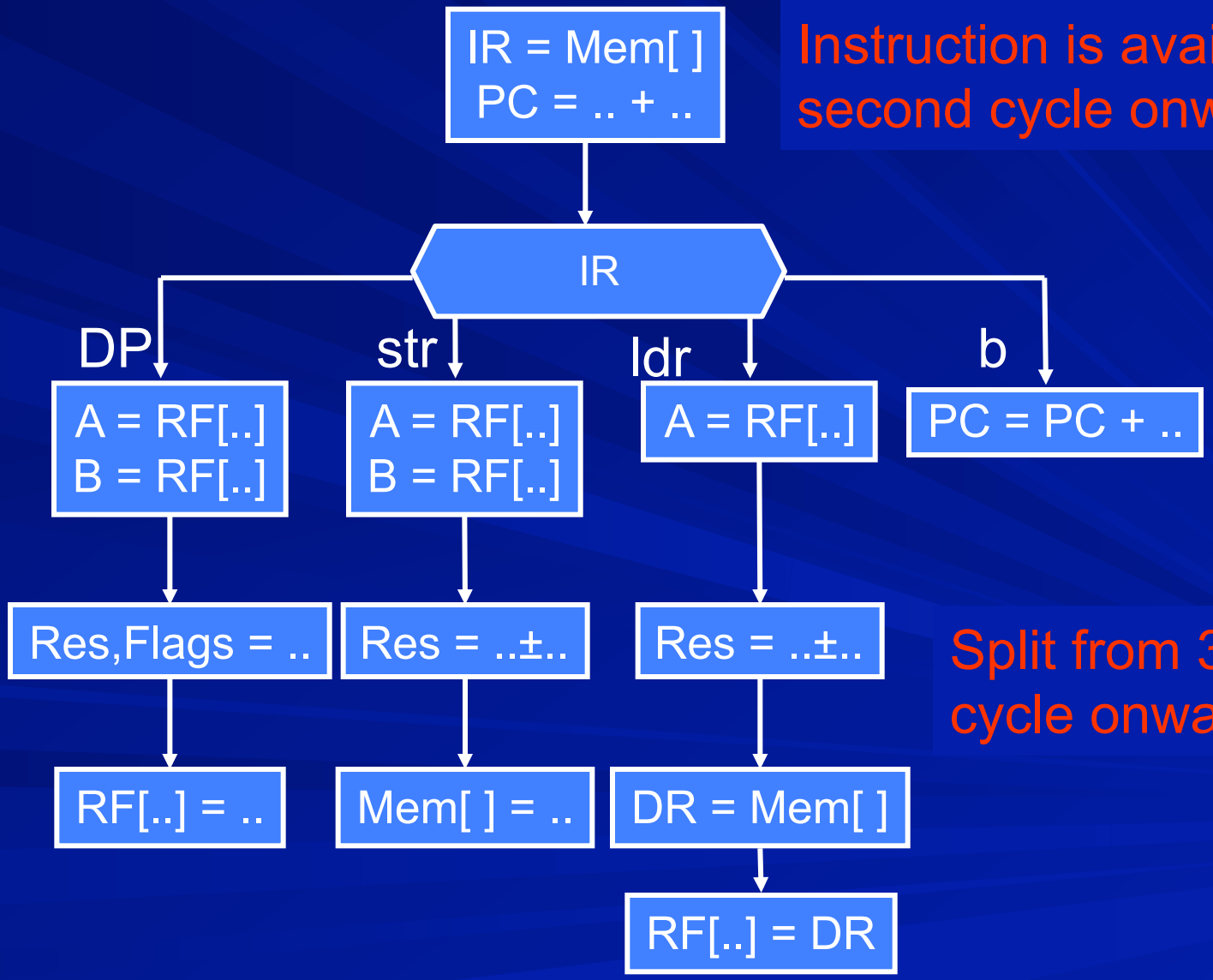
Break Instruction Execution into Cycles: b instruction



Put cycle sequences together



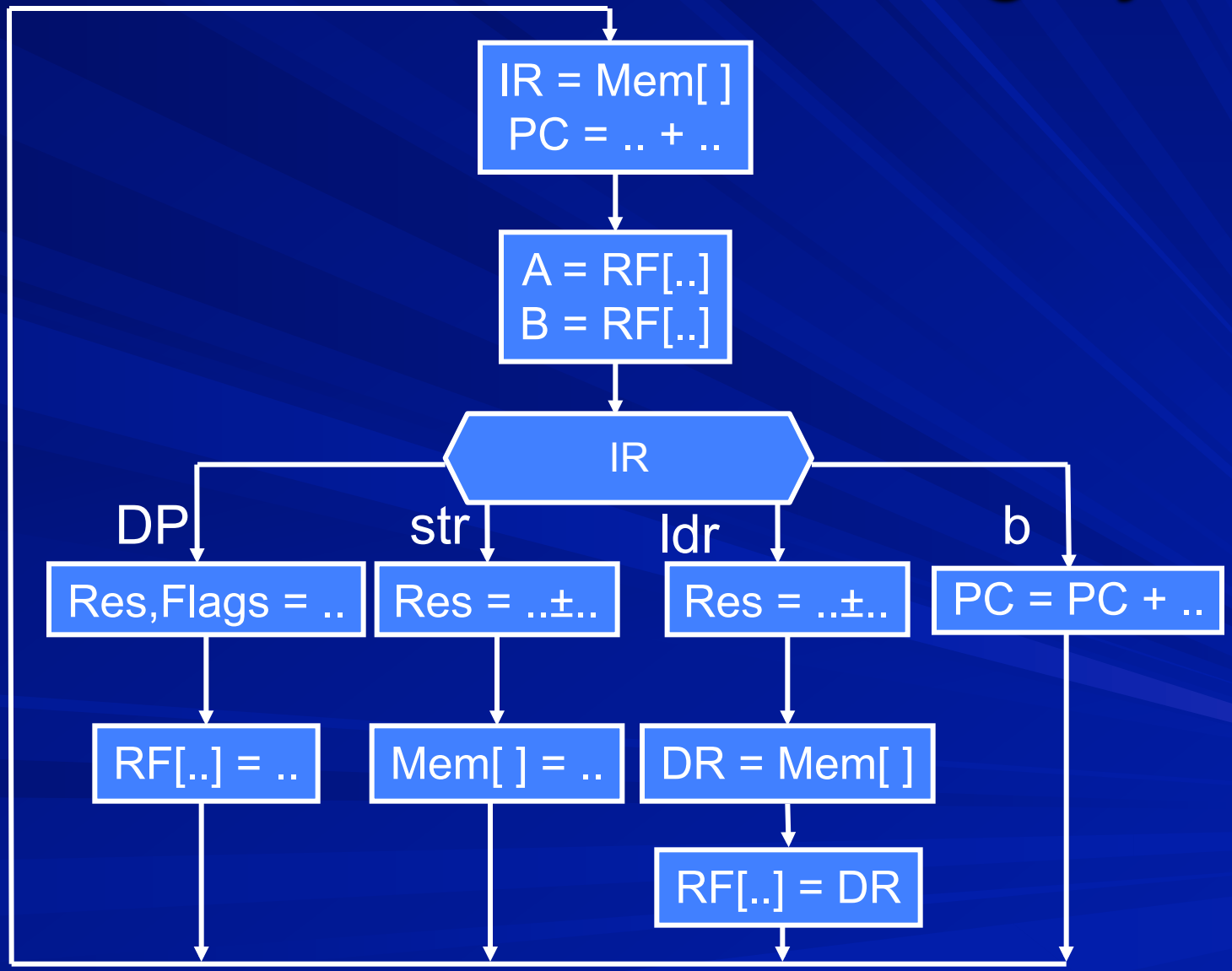
After merging fetch cycle



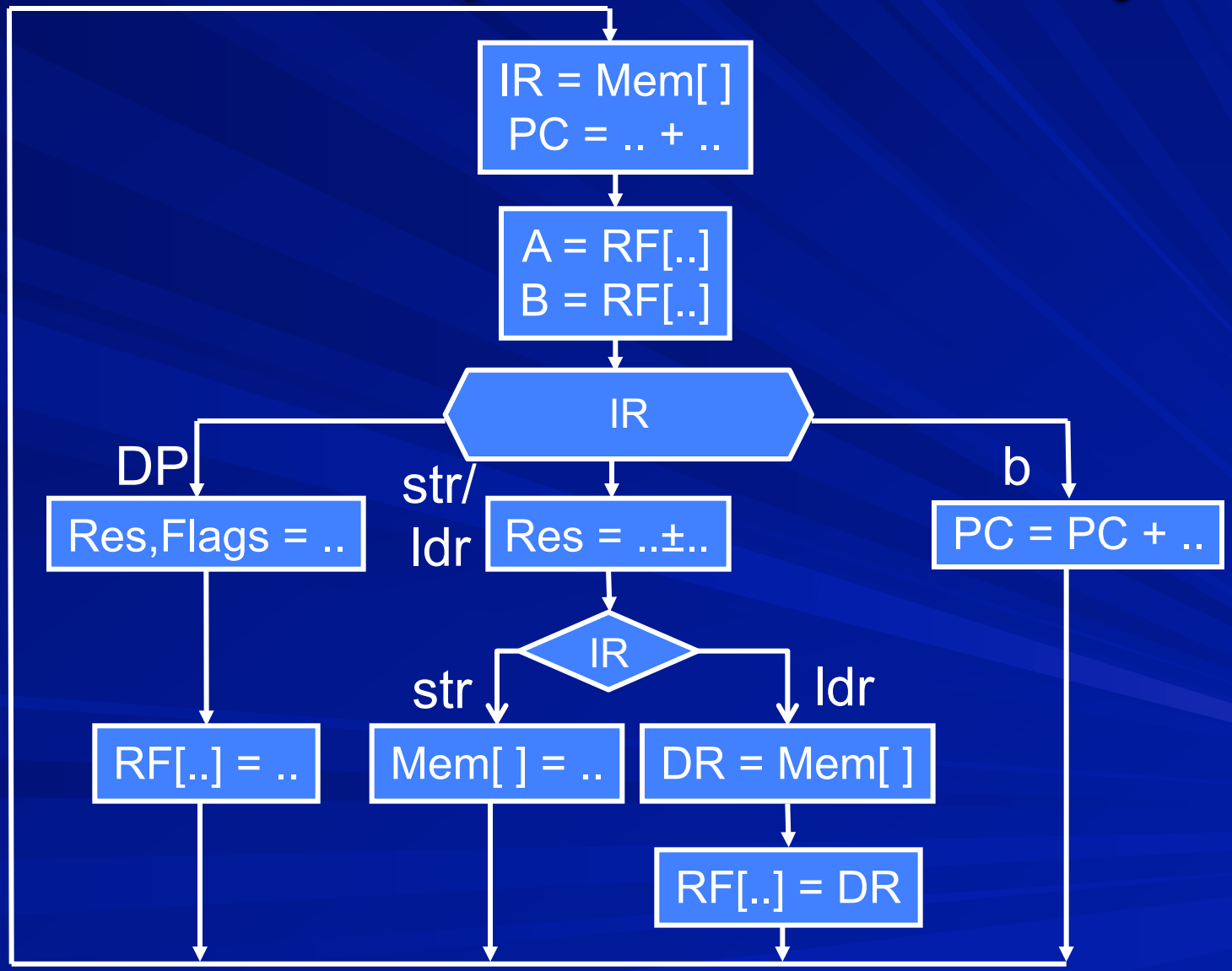
Instruction is available second cycle onwards.

Split from 3rd cycle onwards.

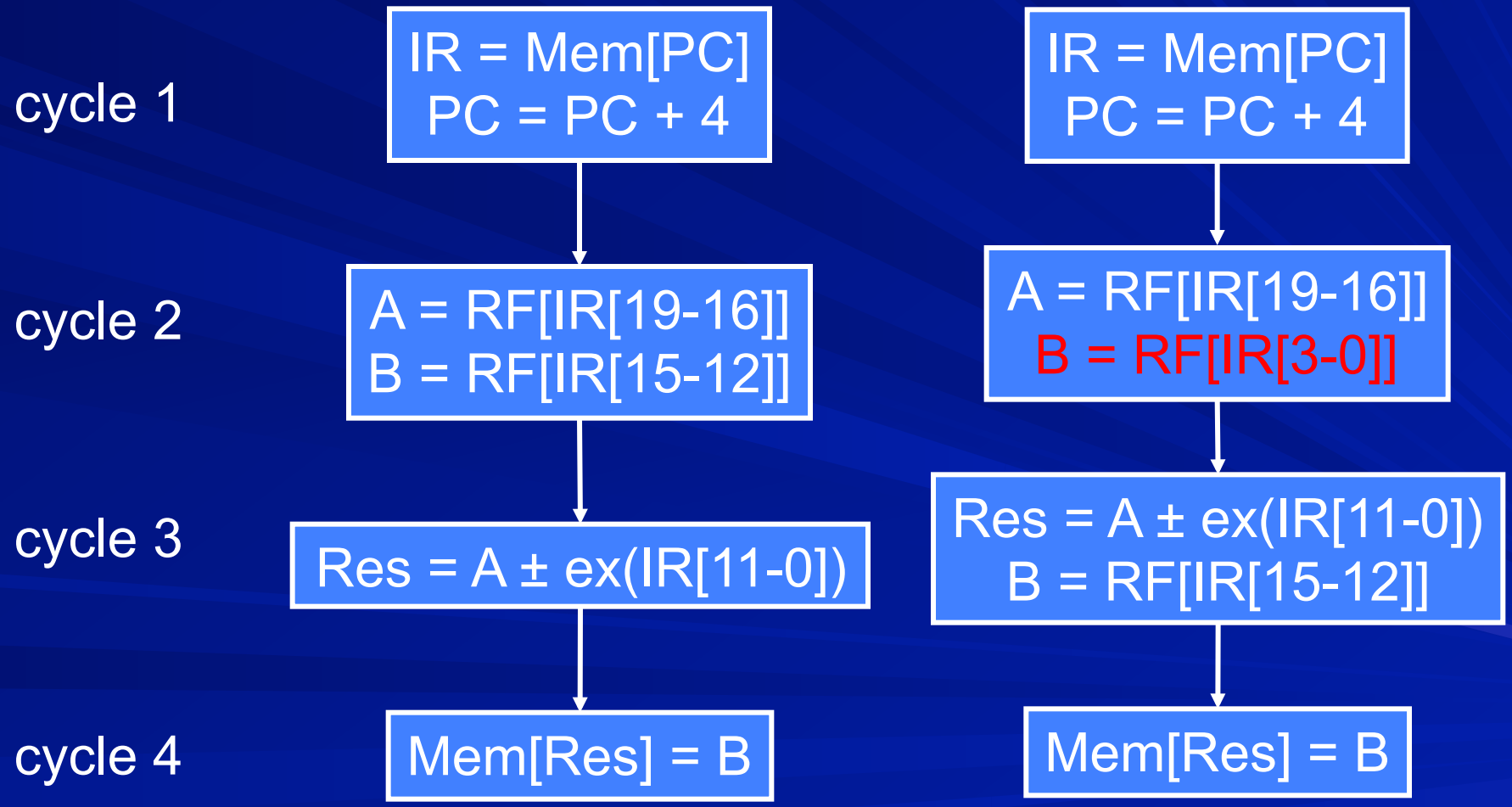
With a common decoding cycle



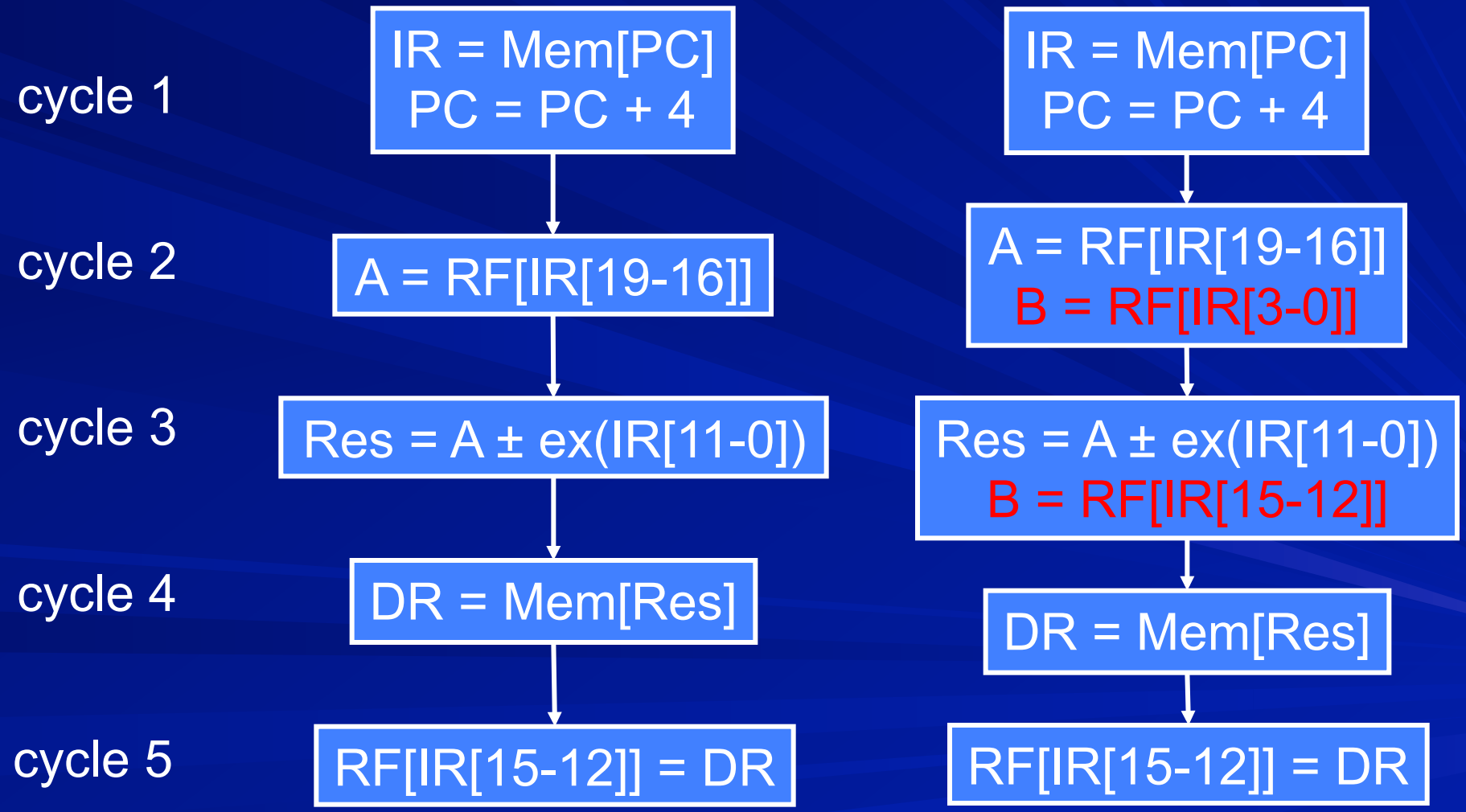
ldr, str can split after third cycle



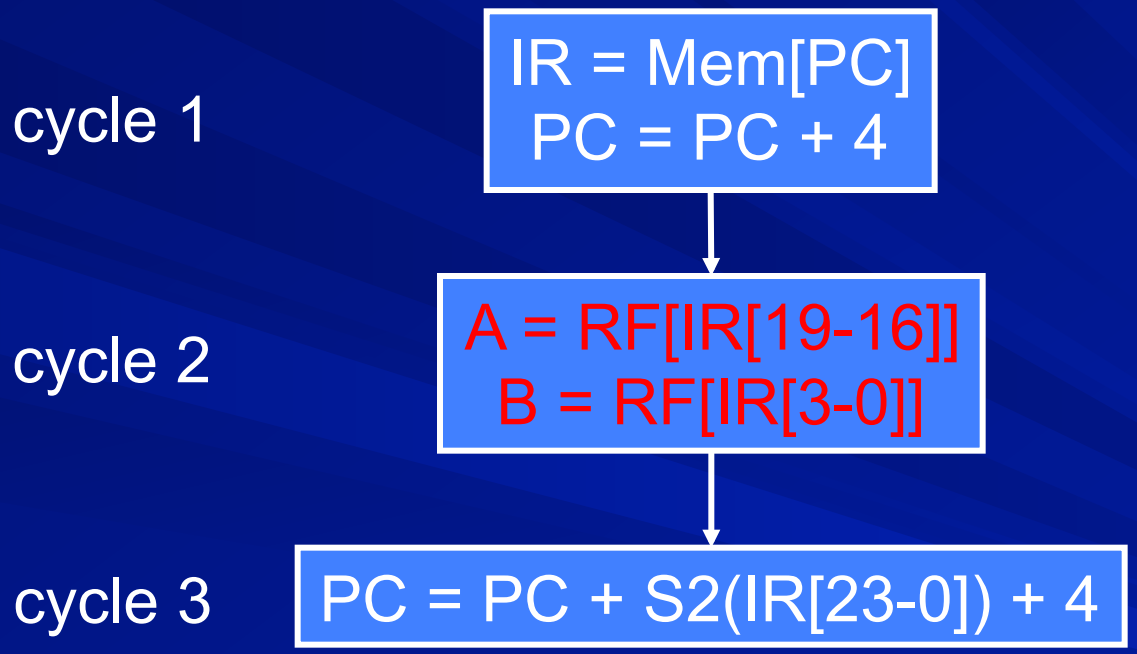
Modified str actions



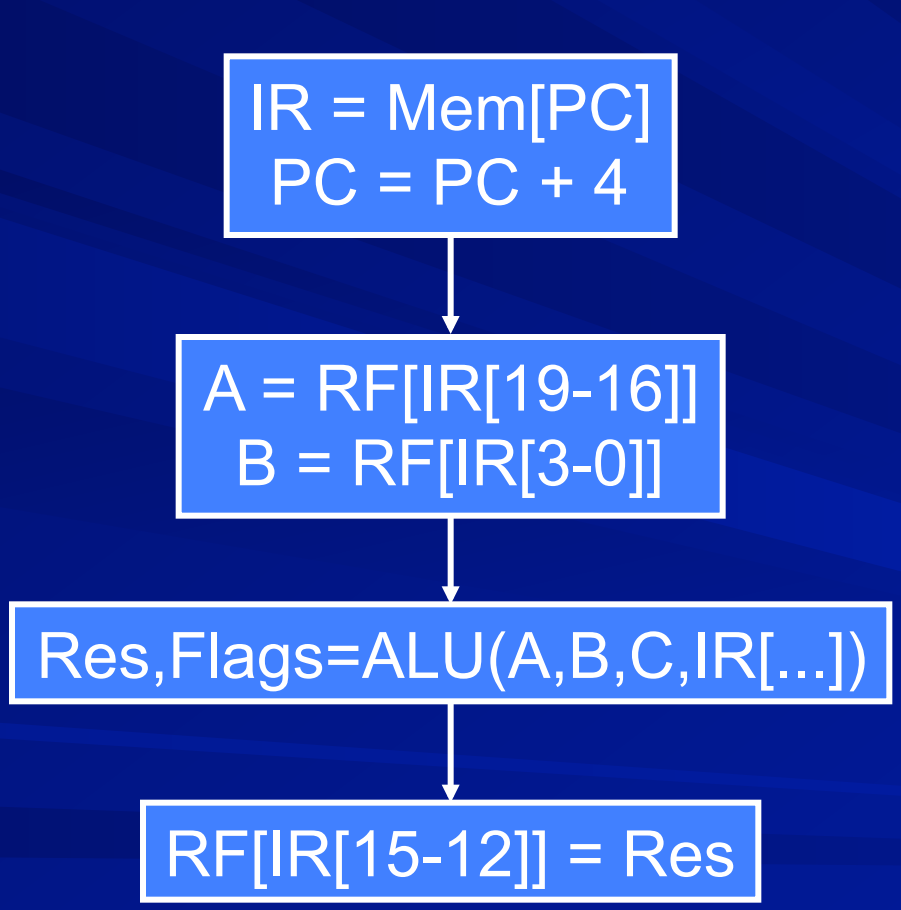
Modified ldr actions

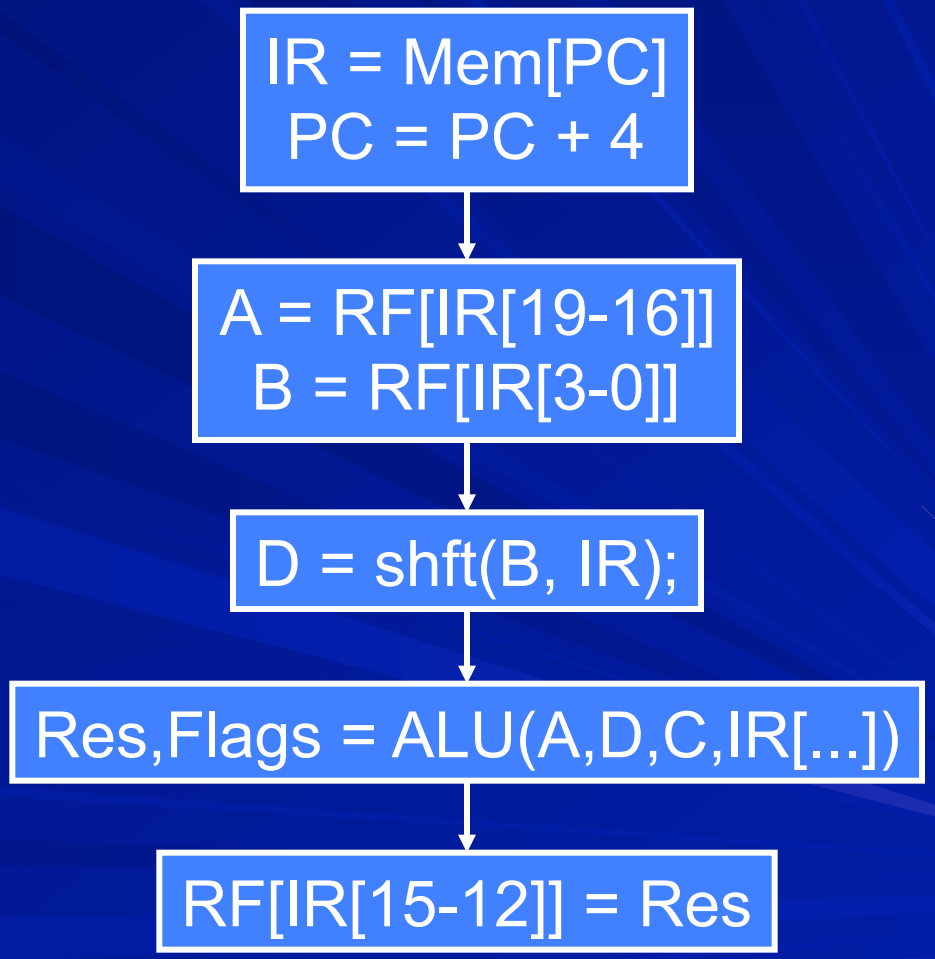


Modified b actions



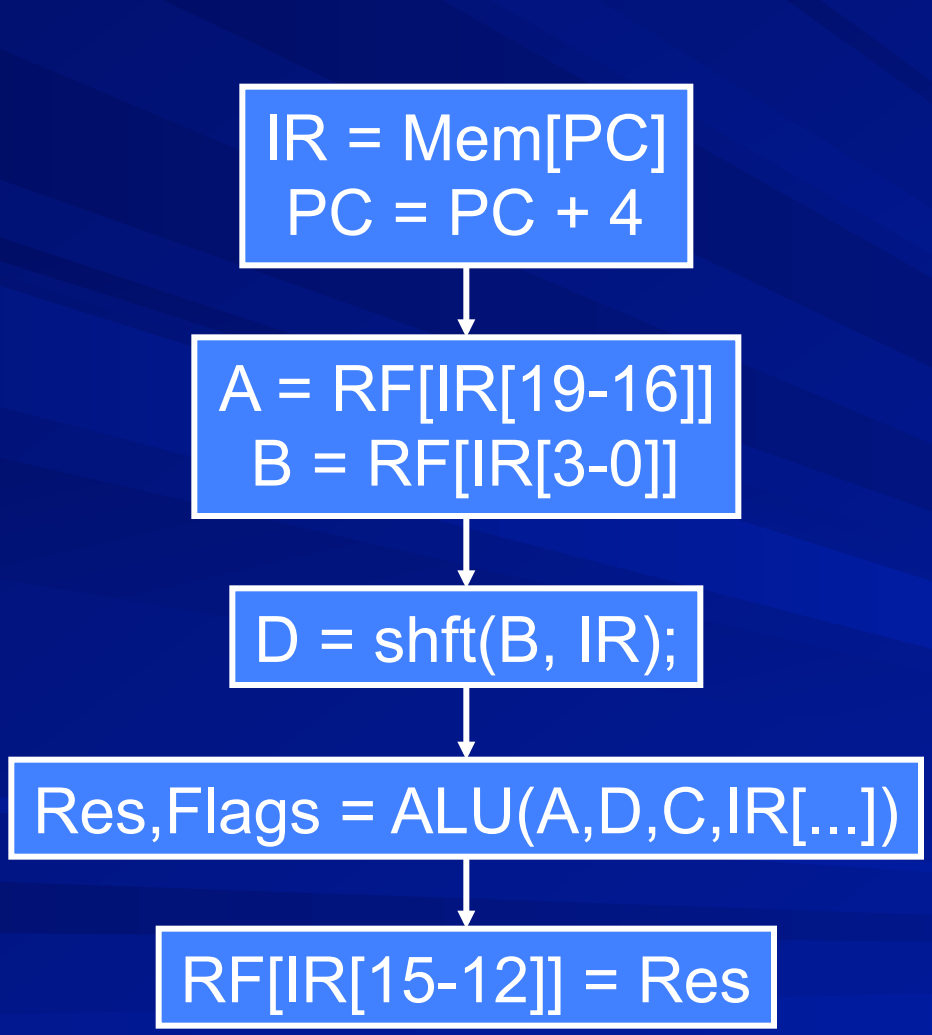
Other forms of DP instructions



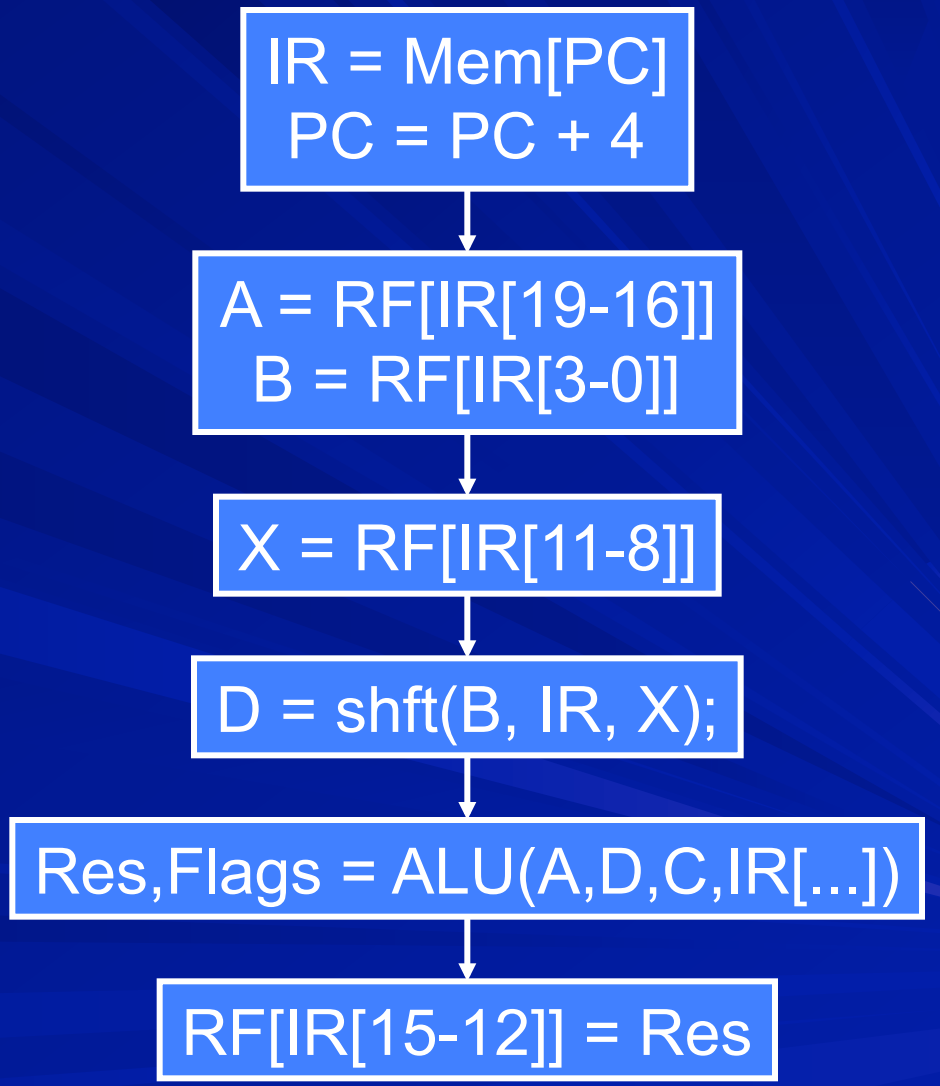


constant shift

Other forms of DP instructions

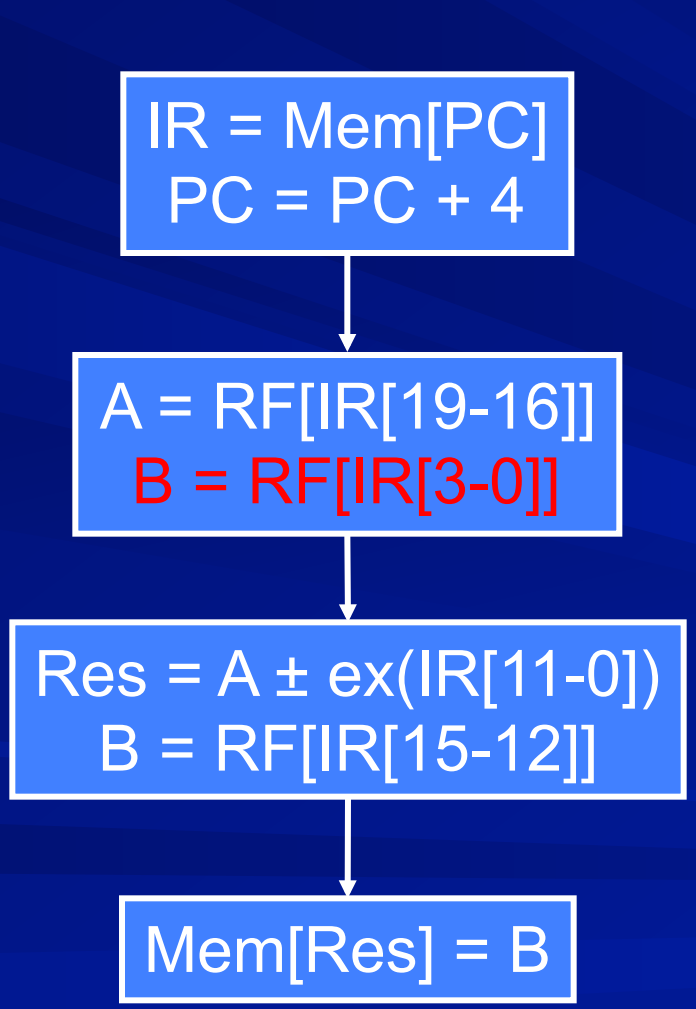


constant shift

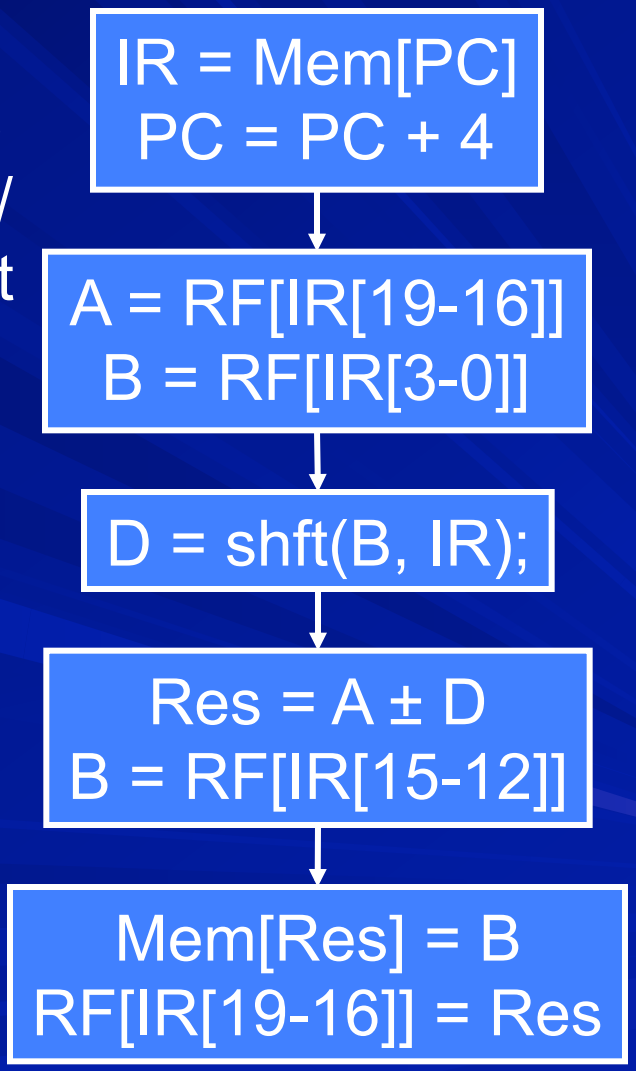


register specified shift

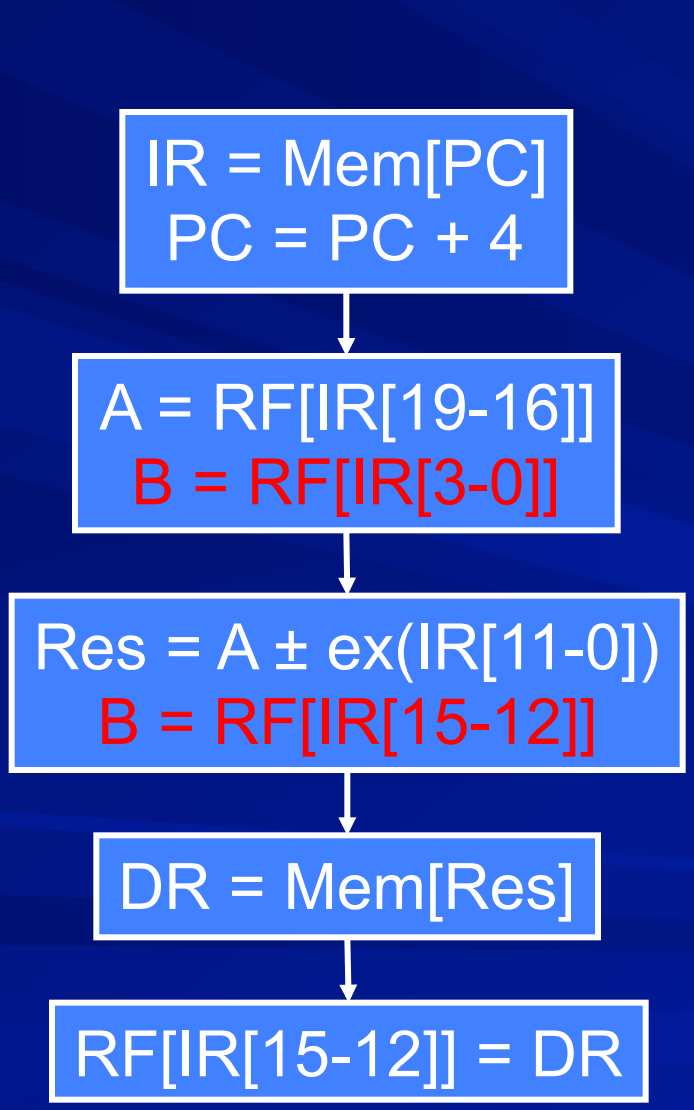
Other forms of str instruction



auto (pre) increment/ decrement register offset



Other forms of ldr instruction



auto (pre) increment/ decrement register offset