Lecture 6: Microcontrolled Multi-Cycle CPU Implementation

Jangwoo Kim (Seoul National University)
jangwoo@snu.ac.kr

Announcement #1

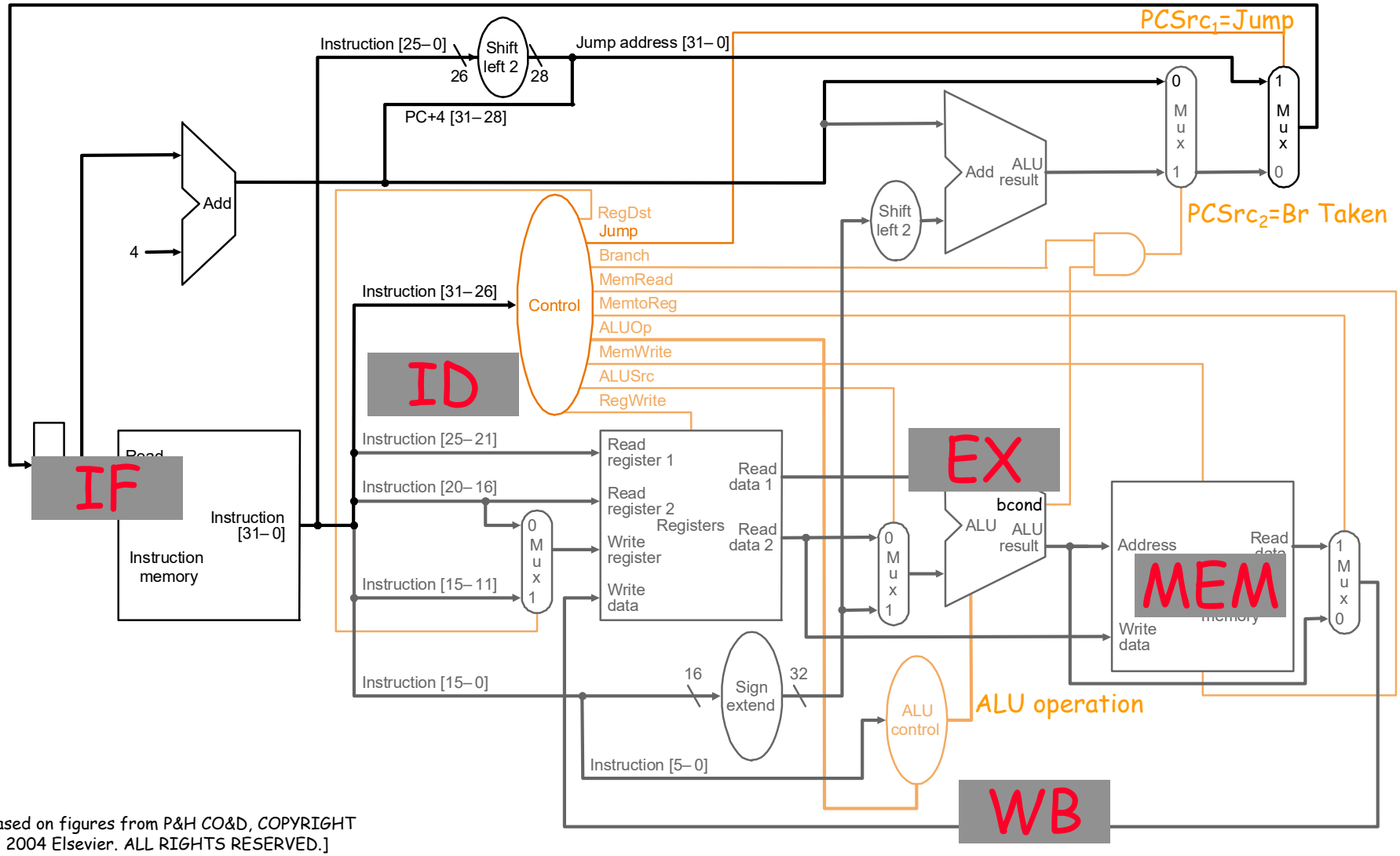
- **Final Exam**
  - 4/20 (Fri), 4/23 (Mon), 4/24 (Tues), or 4/25 (Wed)…?
    - 6:30pm to as long as it takes..
- Homework #2
  - To be posted soon.
  - Due: 4/12
- Q&A board
  - Please appreciate your TAs!
    - Super-busy graduate students are babysitting you!

Single-Cycle Implementation

- Matches naturally with the sequential and atomic semantics inherent to most ISAs
  - Instantiate programmer-visible state one-for-one
  - Map instructions to a combinational next-state logic
- But, contrived and inefficient
  - All instructions run **as slow as the slowest instruction**
    - Which one?
  - Must provide the worst-case combinational resource in parallel as required by any instruction
- Not necessarily the simplest way to implement an ISA

Gets much worse for a CISC ISA

Single-Cycle Implementation



[Based on figures from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

Single-Cycle Datapath Analysis

- Assume
  - Memory units (read or write): 200 ps (=0.2 nano-sec)
  - ALU (add op) : 100 ps
  - Register file (read or write): 50 ps
  - Other combinational logic: 0 ps

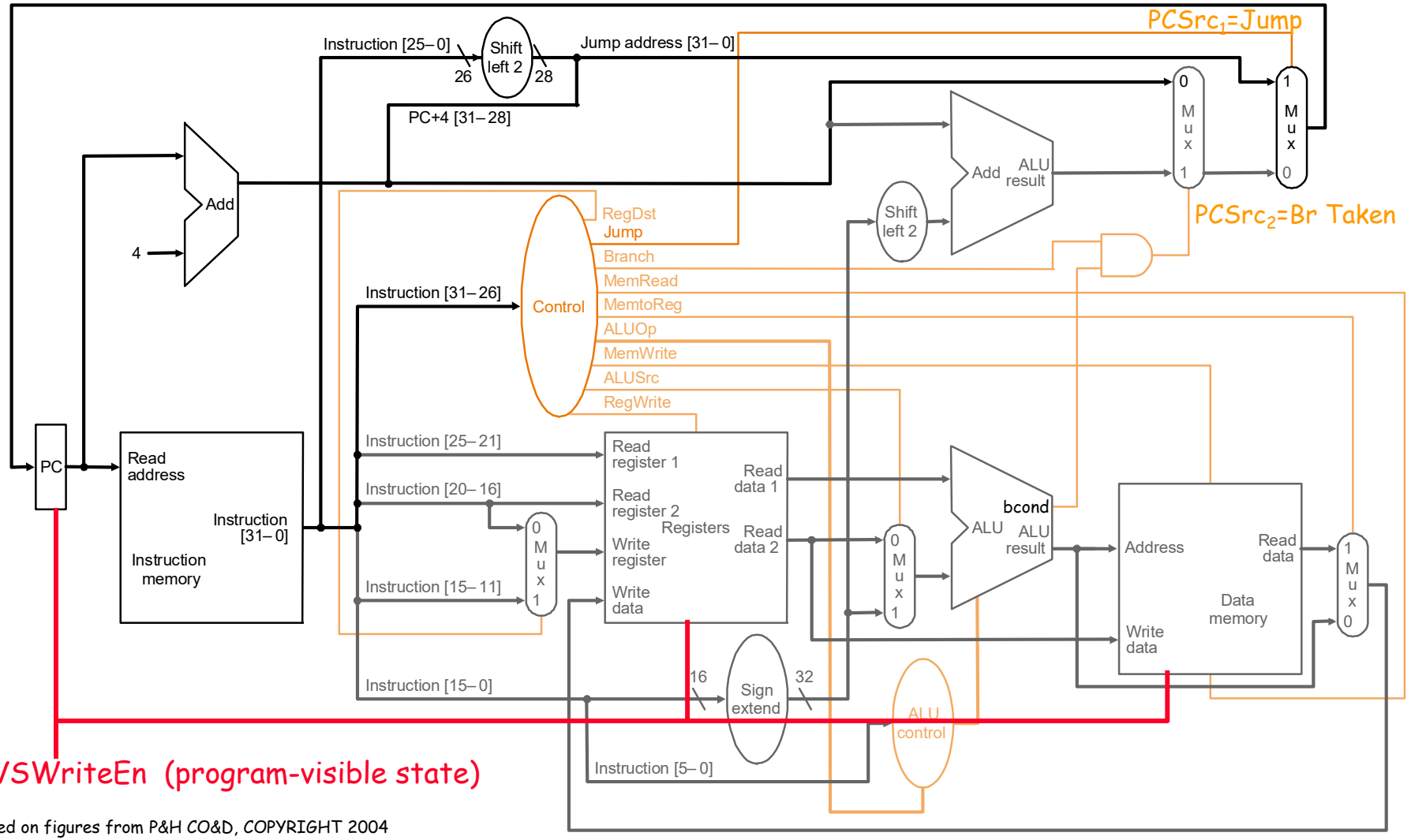
steps	IF	ID	EX	MEM	WB	Delay
resources	mem	RF	ALU	mem	RF	
R-type	200	50	100		50	400
I-type	200	50	100		50	400
LW	200	50	100	200	50	600
SW	200	50	100	200		550
Branch	200	50	100			350
Jump	200					200

Multi-Cycle Implementation: Ver 1.0

- Let's make each instruction type take **only as much time as it needs**
- Idea
  - Run a 50ps clock
  - Let each instruction type take as many clock cycles as needed
  - Programmer-visible state only updates **at the end of an instruction's cycle-sequence**
  - An instruction's effect is still purely combinational from PVS (program visible state) to PVS

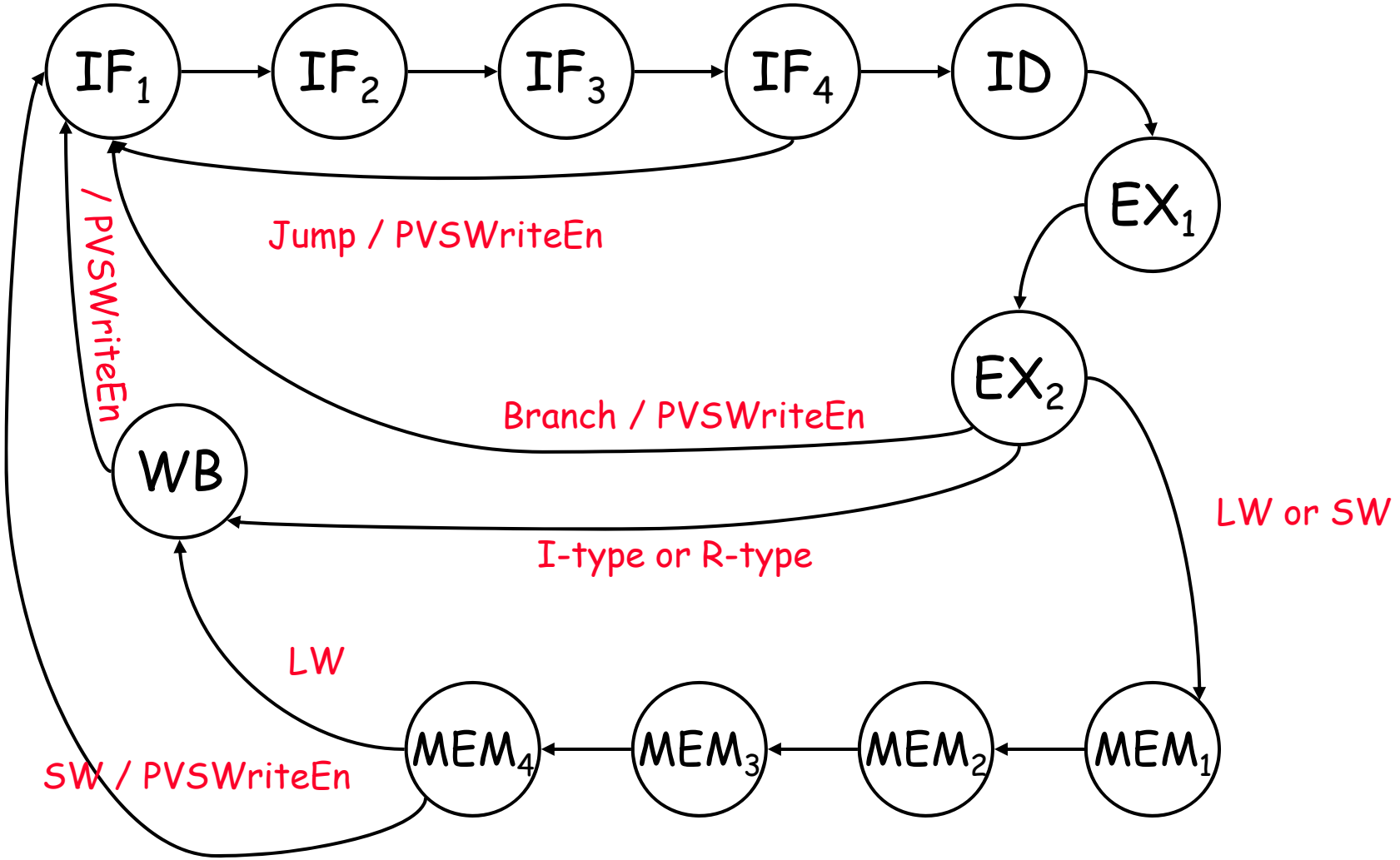
A more realistic alternative to the "variable-length" clock design in the textbook

Multi-Cycle Datapath: Ver 1.0



Sequential Control: Ver 1.0

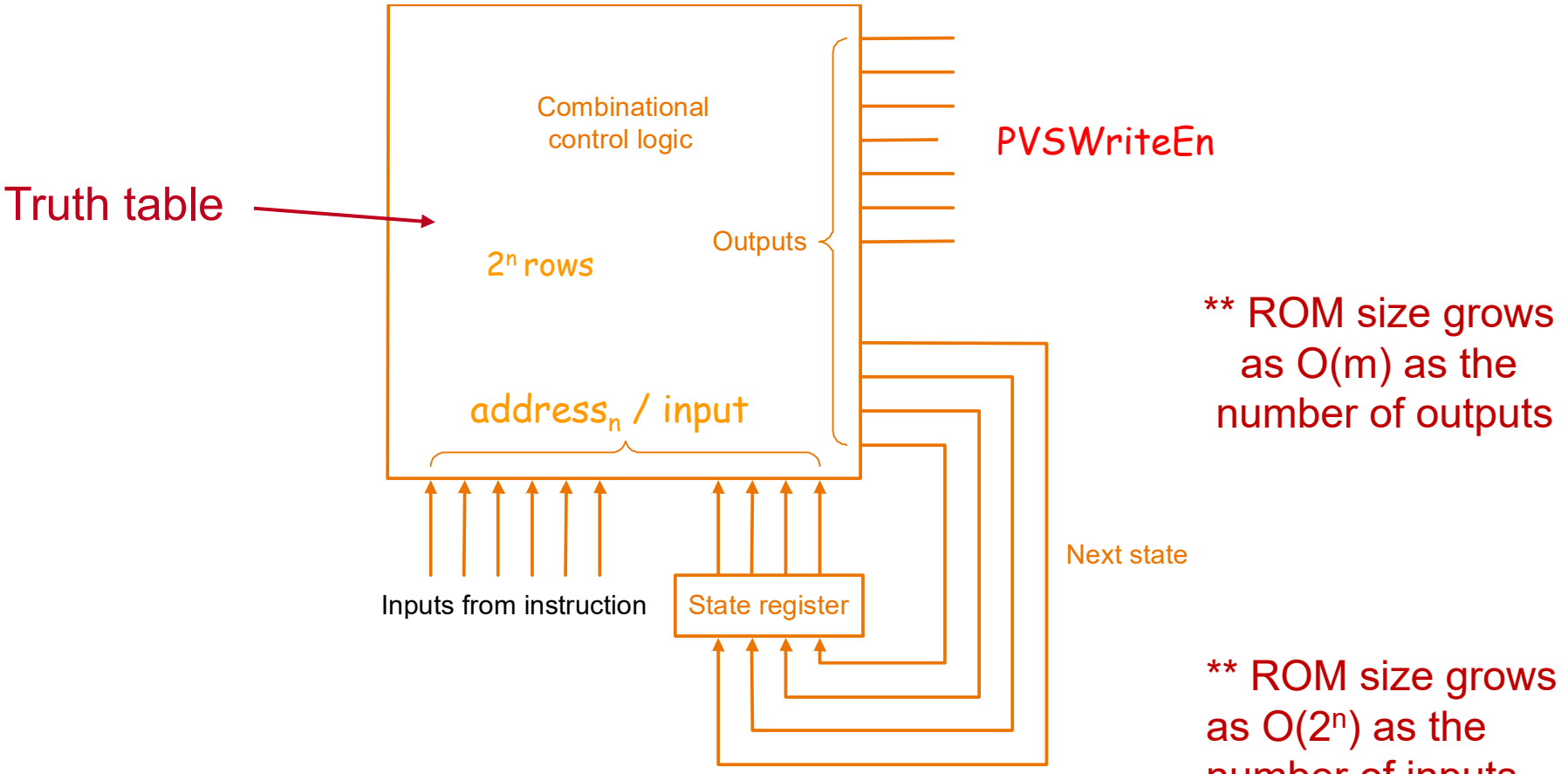
Mealy FSM



What about the rest of the control signals?

MicroSequencer: Ver 1.0

- ROM as a combinational logic lookup table to make FSM



** ROM size grows as O(m) as the number of outputs

** ROM size grows as $O(2^n)$ as the number of inputs

Microcoding: Ver 1.0

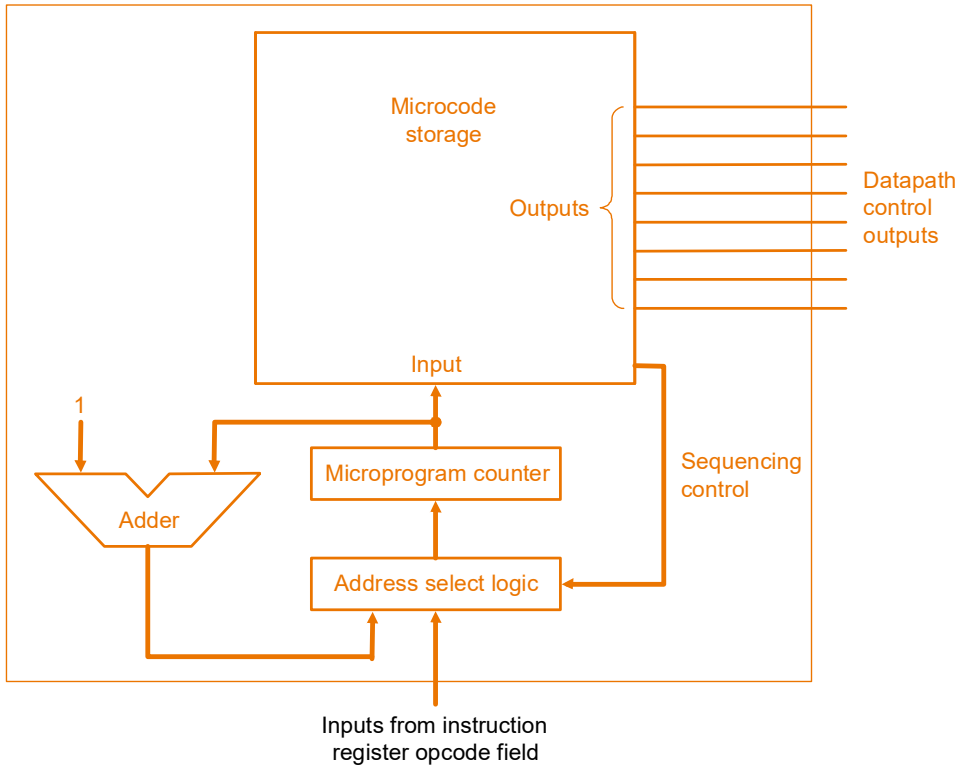
state label	contrl flow	conditional targets				
		R/I-type	LW	SW	Br	Jump
IF_1	next	-	-	-	-	-
IF_2	next	-	-	-	-	-
IF_3	next	-	-	-	-	-
IF_4	go to	ID	ID	ID	ID	IF_1
ID	next	-	-	-	-	
EX_1	next	-	-	-	-	
EX_2	go to	WB	MEM_1	MEM_1	IF_1	
MEM_1	next	-	-	-		
MEM_2	next	-	-	-		
MEM_3	next	-	-	-		
MEM_4	go to	-	WB	IF_1		
WB	go to	IF_1	IF_1			

"More systematic approach" to FSM sequencing/control

Micro-code controller

- A small processor for sequencing and control purpose
  - Control states are like μPC
  - μPC indexed into a μprogram ROM to select an μinstruction
  - Well-formed control-flow architecture
  - Fields in the μinstruction maps to control signals

    Why not also support μprogram-visible states?



- Smart microcontrollers have been built already!
  - Some μcontrollers supported full-scale ISAs
  - Using μISAs for CISC machines inspired RISC ISA

Performance Analysis

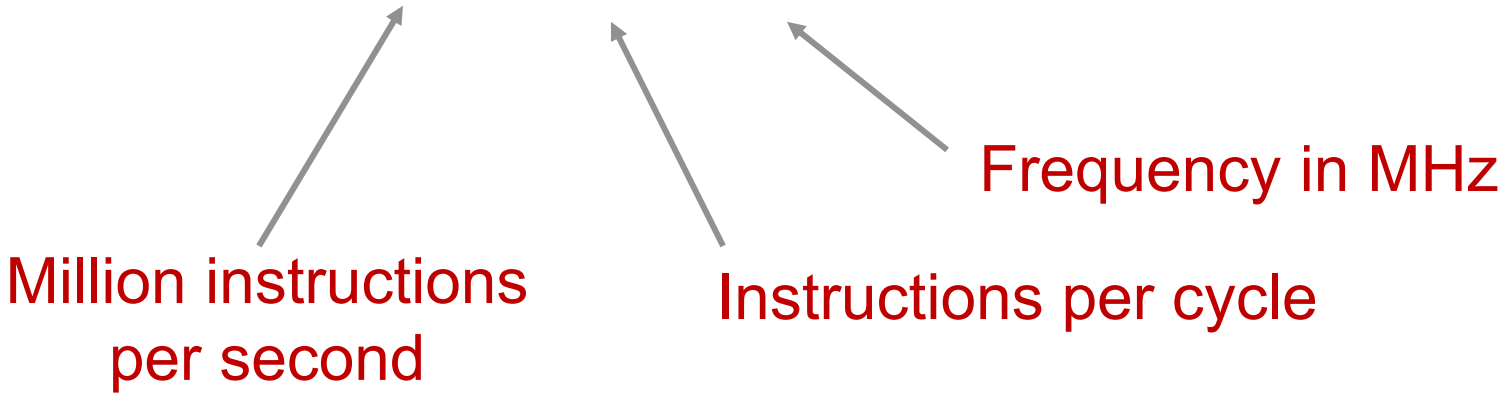
- $T_{wall\text{-}clock}$ = # of instructions × CPI × T_{clk}
  - "# of instruction" is fixed for a given ISA and application

- For a fixed ISA and application, we can compare

$$T_{avg\text{-}inst} = CPI \times T_{clk}$$

or

$$MIPS = IPC \times f_{clk}$$



Performance Analysis

- Single-Cycle Implementation (MIPS = IPC × f_{clk})

  1 × 1,667MHz = 1667 MIPS // 600ps clock period

If 1 clock == 12 μclock

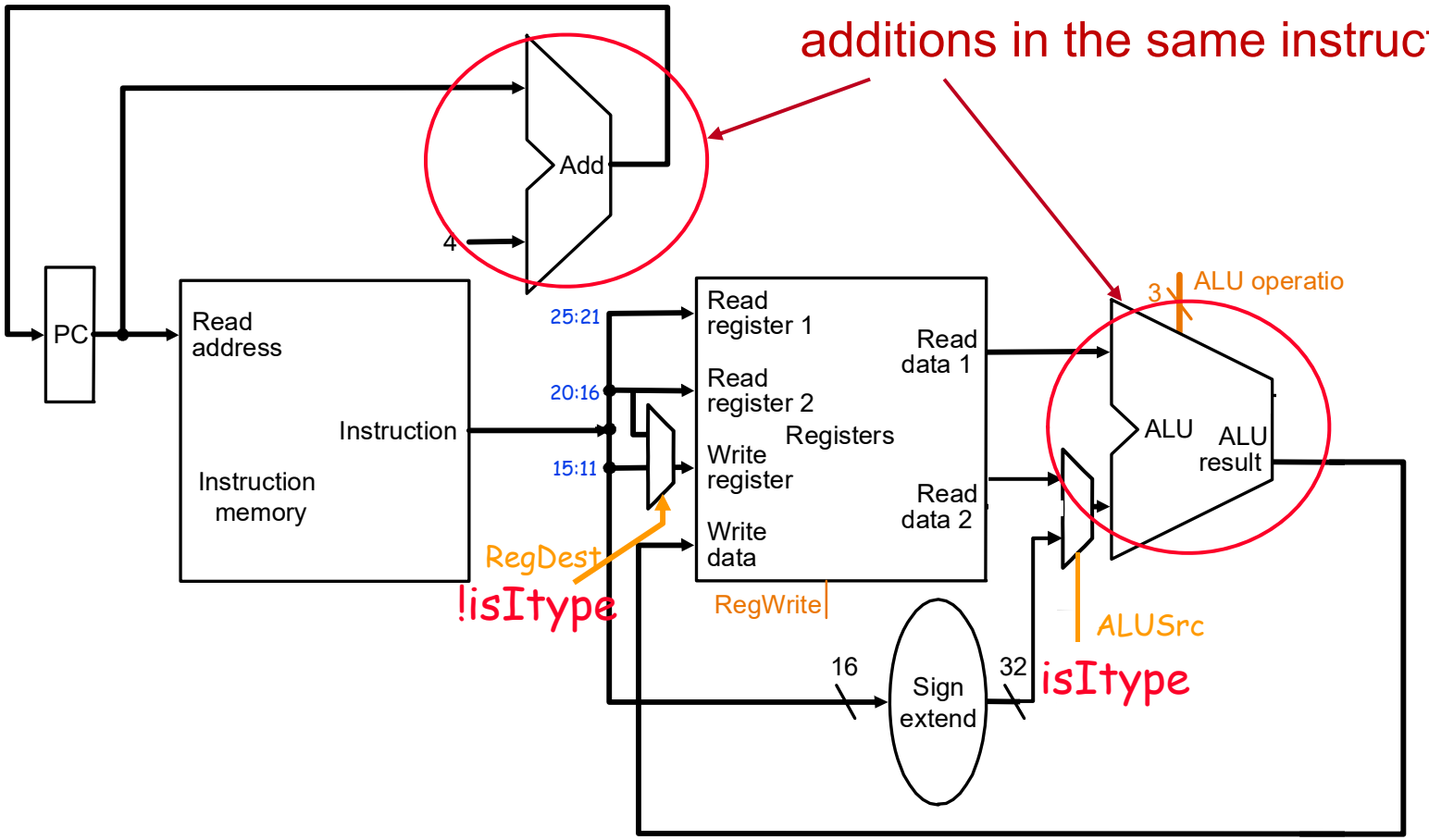
- Multi-Cycle Implementation = 2235 MIPS !!

  IPC_{avg} × 20,000 MHz What is average IPC?

- Assume: 25% LW, 10% SW, 45% ALU, 15% Branch and 5% Jumps
  - Weighted arithmetic mean (WAM) of CPI
    - 0.25*12+0.1*11+0.45*8+0.15*7+0.05*4 = 8.95
  - **Weighted harmonic mean (WHM) of IPC** = 1/CPI= 0.1117

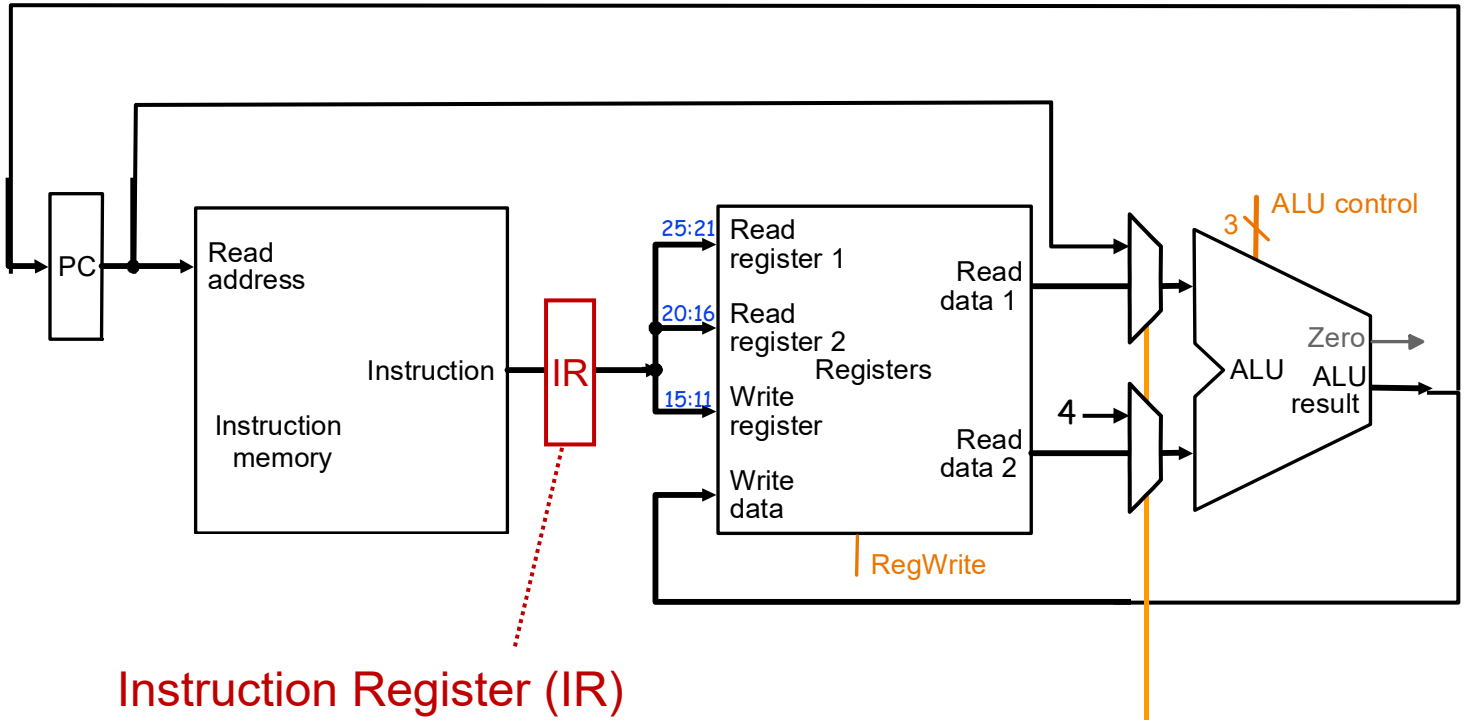
Reducing Datapath by Resource Reuse

How to reuse the same adder for both additions in the same instruction?



Previous example of reuse by mutually exclusive conditions

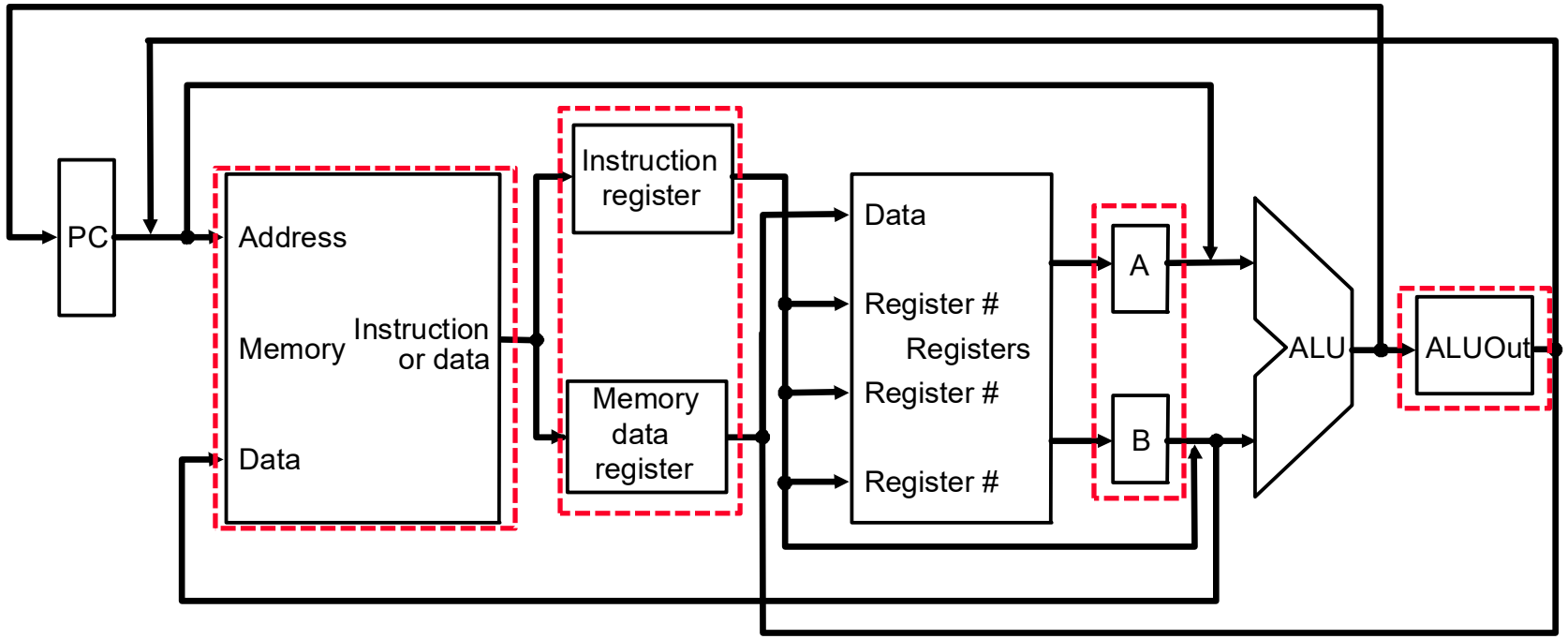
Reducing Datapath (ALU) by Sequential Reuse



Instruction Register (IR)

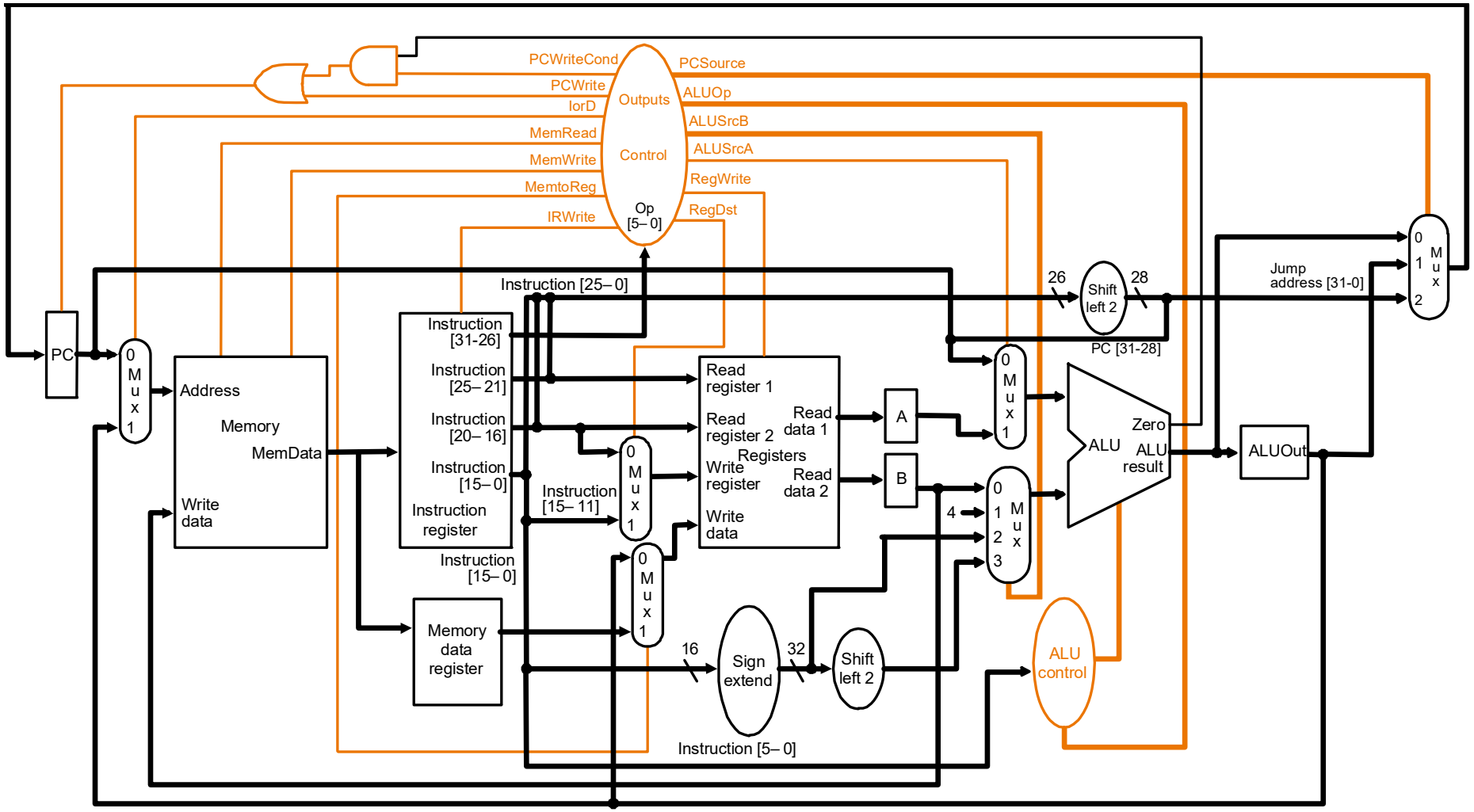
But, must create timing difference between IF/ID and PC=PC+4

Removing Redundancies



We can use single memory & register by creating **timing differences** with proper controls.

Control Points



New Sequential Control Signals

	When De-asserted	When asserted
ALUSrcA	1st ALU input from **PC**	1st ALU input from 1st **RF** read port (latched in **A**)
IorD	**PC** supplies memory address (of instruction)	**ALUOut** supplies memory address (of data)
IRWrite	**IR** latching disabled	**IR** latching enabled
PCWrite	no effect	**PC** latching enabled unconditionally
PCWriteCond	no effect	**PC** latching enabled only if branch condition is satisfied

*When both PCWrite and PCWriteCond are de-asserted, **PC** latching is disabled*

New Sequential Control Signals

signal		effect
ALUSrcB[1:0]	00	2nd ALU input from 2nd **RF** read port (latched in **B**)
	01	2nd ALU input is 4 (for PC increment)
	10	2nd ALU input is sign-extended **"IR[15:0]"** (for # data)
	11	2nd ALU input is sign-extended **"IR[15:0],00"** (for branching)
PCSource[1:0]	00	next **PC** from ALU
	01	next **PC** from **ALUOut**
	10	next PC from **IR** (jump target)

Old Control Signals (similar to single-cycle)

	When De-asserted	When asserted
RegDest	**RF** write select according to **IR[20:16]**	**RF** write select according to **IR[15:11]**
RegWrite	**RF** write disabled	**RF** write enabled
MemRead	Memory read disabled	Memory read port return load value
MemWrite	Memory write disabled	Memory write enabled
MemtoReg	Steer ALU result (latched in **ALUOut**) to **RF** write port	steer memory load result (latched in **MDR**) to **RF** write port

Synchronous Register Transfers

- Synchronous state with latch enabled by control
  - **PC**, **IR**, **RF**, **MEM**
- Synchronous state that always latch
  - **A**, **B**, **ALUOut**, **MDR**
- Now we can explain all possible combinational "**Register Transfers**" in the datapath!
- For example starting from **PC**
  - **IR ⇐ MEM[PC]** // flow from PC to IR
    requires IorD=0, MemRead=1, IRWrite=1
  - **PC ⇐ PC,JumpTarget** // flow from IR to PC
    requires PCWrite=1, PCSource=2'b10
  - **PC ⇐ PC + ALUSrcB** requires ?
  - **MDR ← MEM[PC]** requires ?
  - **ALUOut ← PC + ALUSrcB** requires ?

    ……

Useful Register Transfers

- PC ⇐ PC+4
- PC ⇐ ALUOut (if PCWriteCond is asserted)
- PC ⇐ PC[31:28],IR[25:0],2'b00
- IR ⇐ MEM[PC]
- A ← RF[IR[25:21]]
- B ← RF[IR[20:16]]
- ALUOut ← A + B
- ALUOut ← A + sign-extend (IR[15:0])
- ALUOut ← PC + (sign-extend (IR[15:0]) <<2)
- MDR ← MEM[ALUOut]
- MEM[ALUOut] ⇐ B
- RF[IR[15:11]]⇐ALUOut,
- RF[IR[20:16]]⇐ALUOut
- RF[IR[20:16]] ⇐ MDR

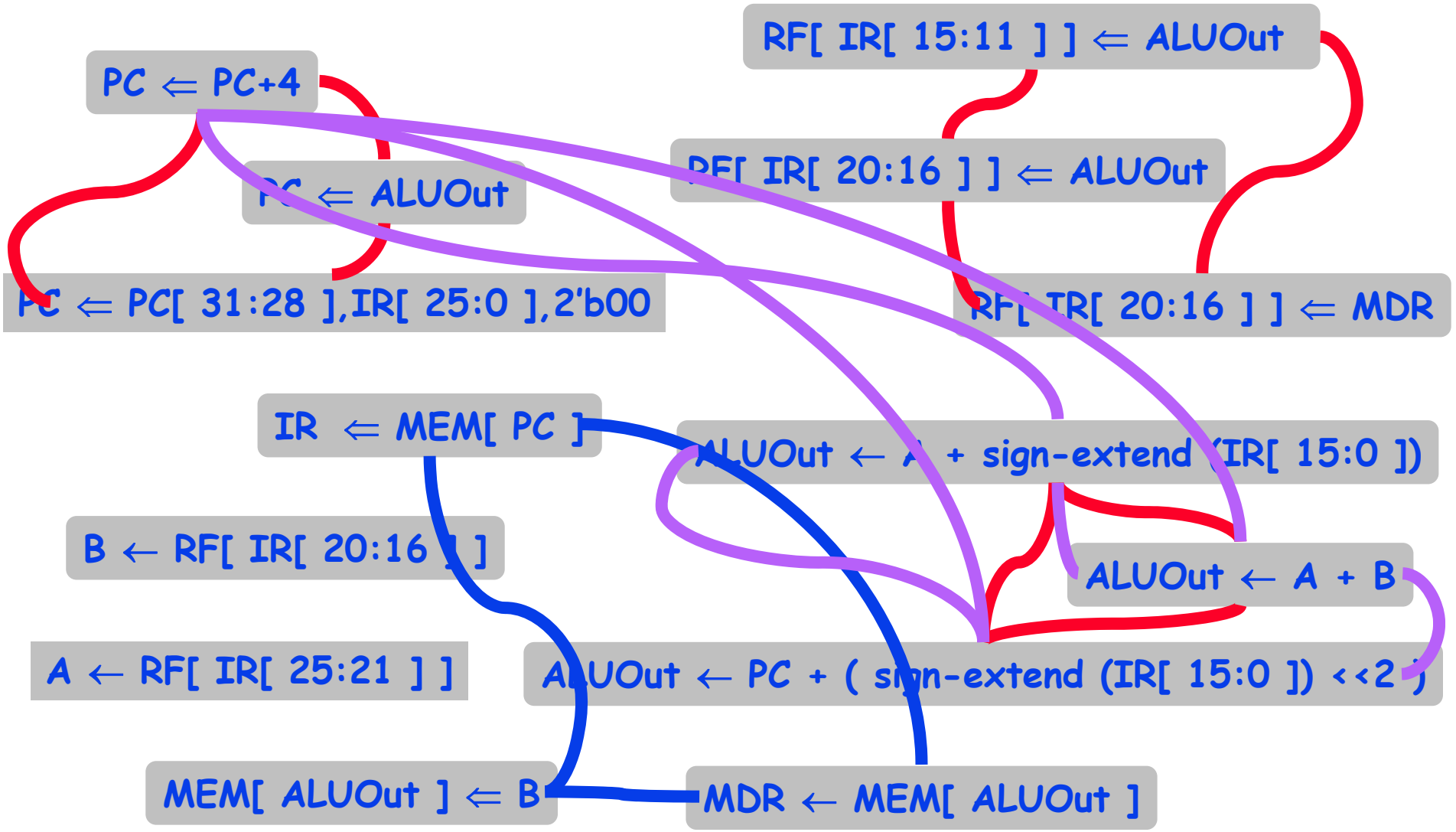
Which control signals to assert/de-assert ?

RT Sequencing: R-Type ALU

- IF
  - IR ⇐ MEM[PC]
  - PC ⇐ PC+4
- ID
  - A ← RF[IR[25:21]]
  - B ← RF[IR[20:16]]
- EX
  - ALUOut ← A + B
- MEM
  -
- WB
  - RF[IR[15:11]] ⇐ ALUOut

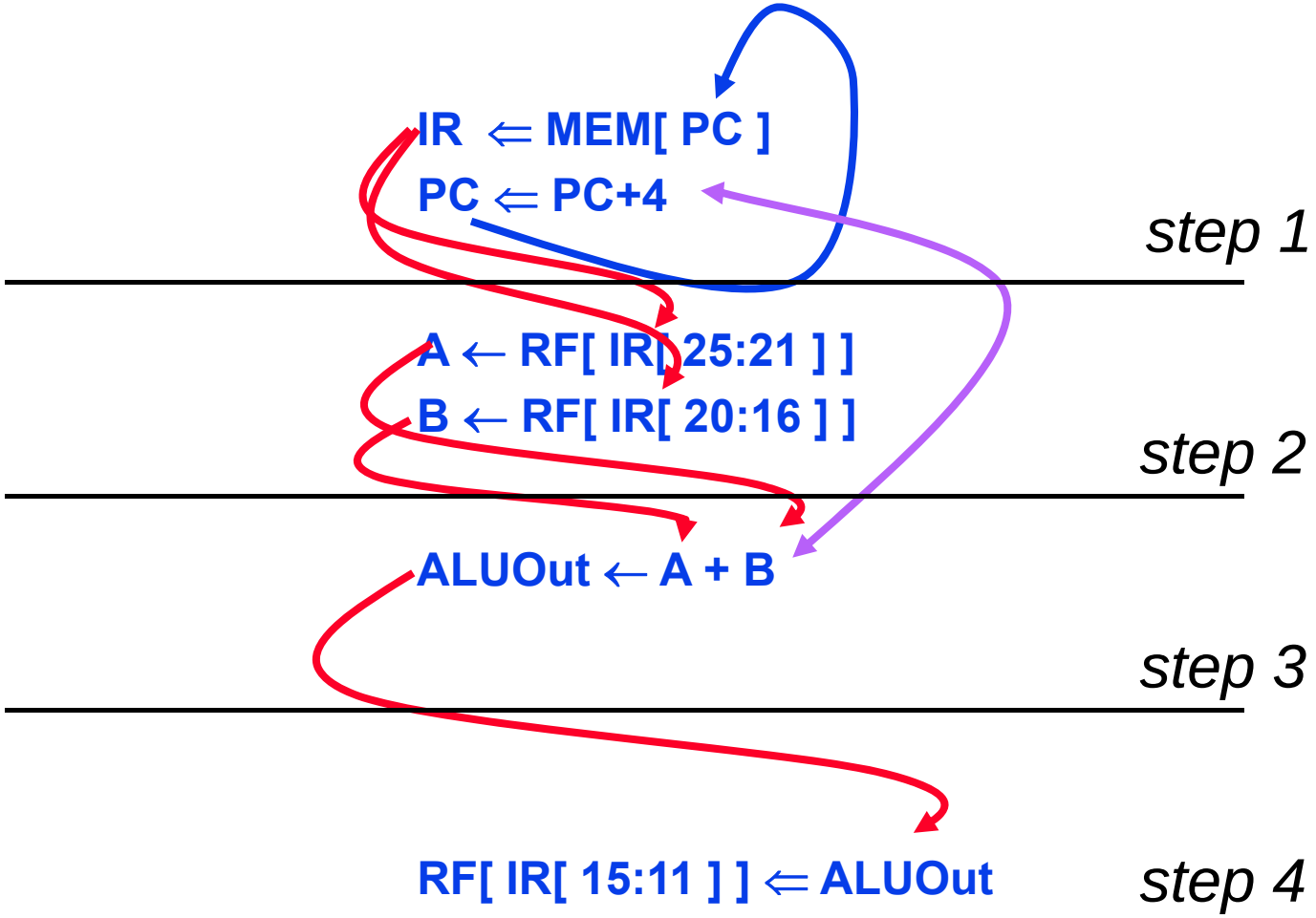
```
if MEM[PC] == ADD rd rs rt
        GPR[rd] ← GPR[rs] + GPR[rt]
        PC ← PC + 4
```

RT Datapath Conflicts



Can utilize each resource only once per control step (cycle)

RT Sequencing: R-Type ALU



RT Sequencing: LW

- IF
  - IR ⇐ MEM[PC]
  - PC ⇐ PC+4
- ID
  - A ← RF[IR[25:21]]
  - B ← RF[IR[20:16]]
- EX
  - ALUOut ← A + sign-extend (IR[15:0])
- MEM
  - MDR ← MEM[ALUOut]
- WB
  - RF[IR[20:16]] ⇐ MDR

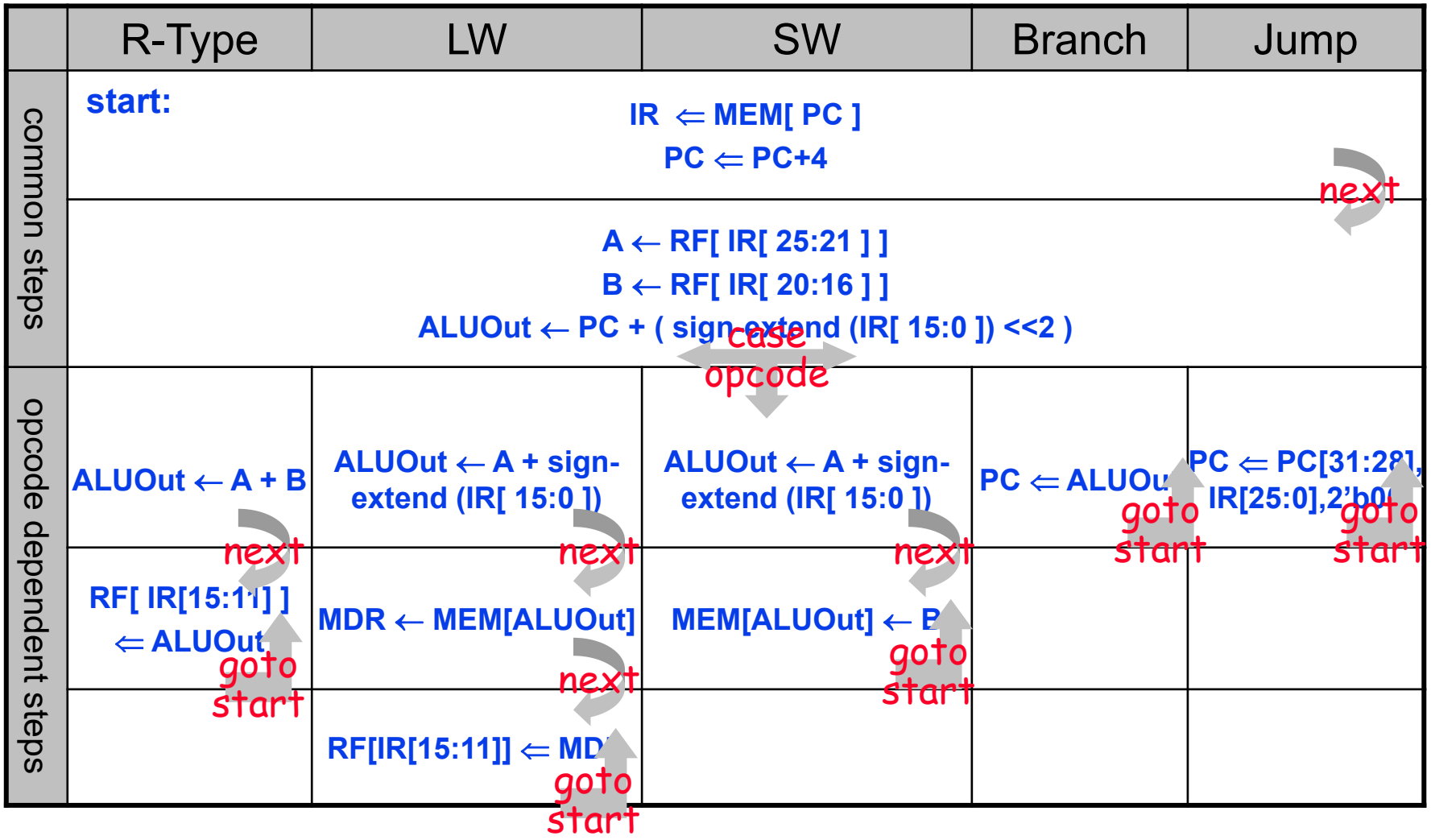
```
if MEM[PC]==LW rt offset16 (base)
      EA = sign-extend(offset) + GPR[base]
      GPR[rt] ← MEM[ translate(EA) ]
      PC ← PC + 4
```

RT Sequencing: Branch

- IF
  - IR ⇐ MEM[PC]
  - PC ⇐ PC+4
- ID
  - A ← RF[IR[25:21]]
  - B ← RF[IR[20:16]]
  - ALUOut←PC+(sign-extend (IR[15:0])<<2)
- EX
  - PC ⇐ ALUOut (only if condition is met)
- MEM
- WB

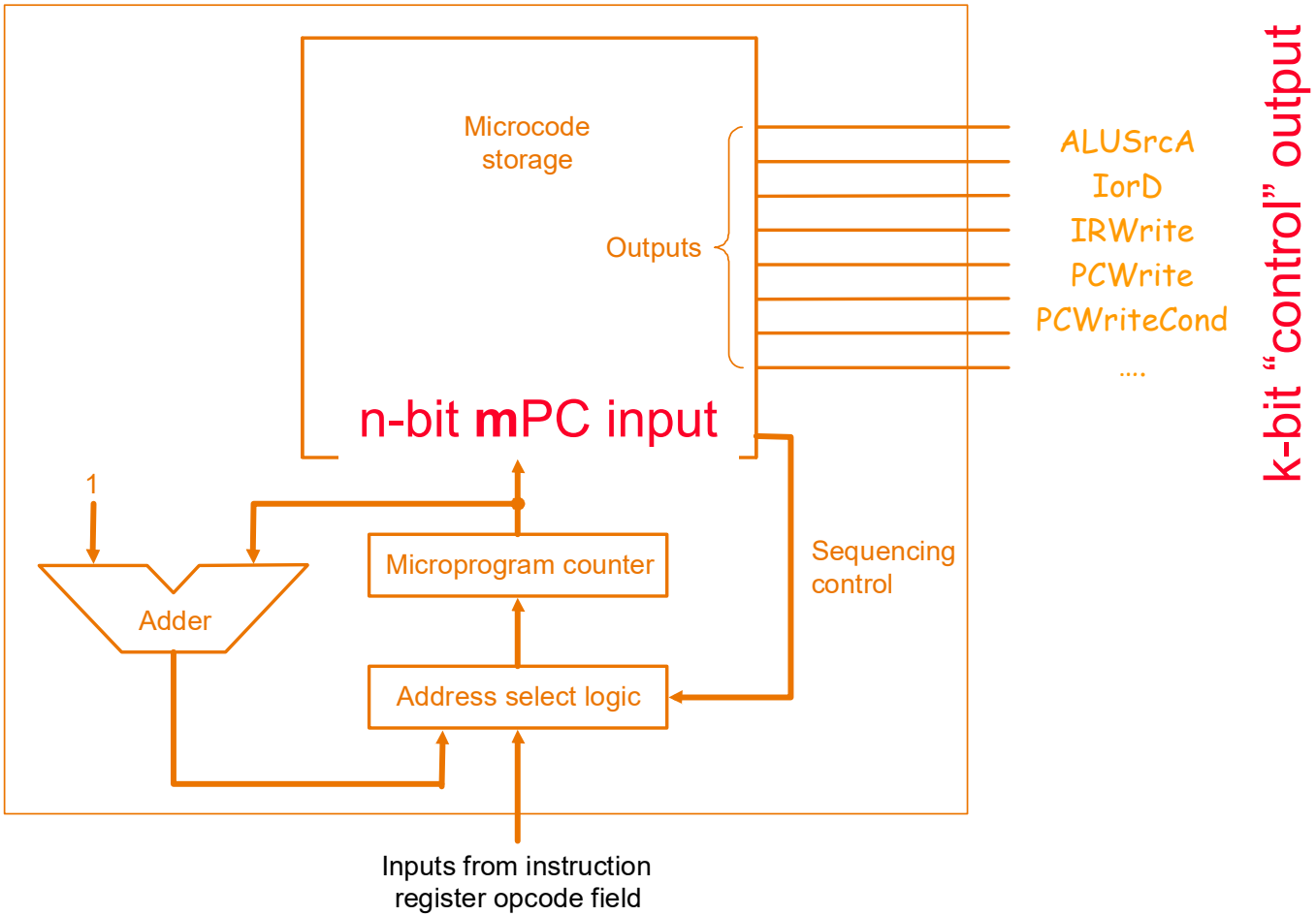
```
if MEM[PC]==BEQ rs rt immediate16
        target = PC + sign-extend(immediate) x 4 +4
        if GPR[rs]==GPR[rt] then     PC ← target
                            else     PC ← PC + 4
```

Combined RT Sequencing



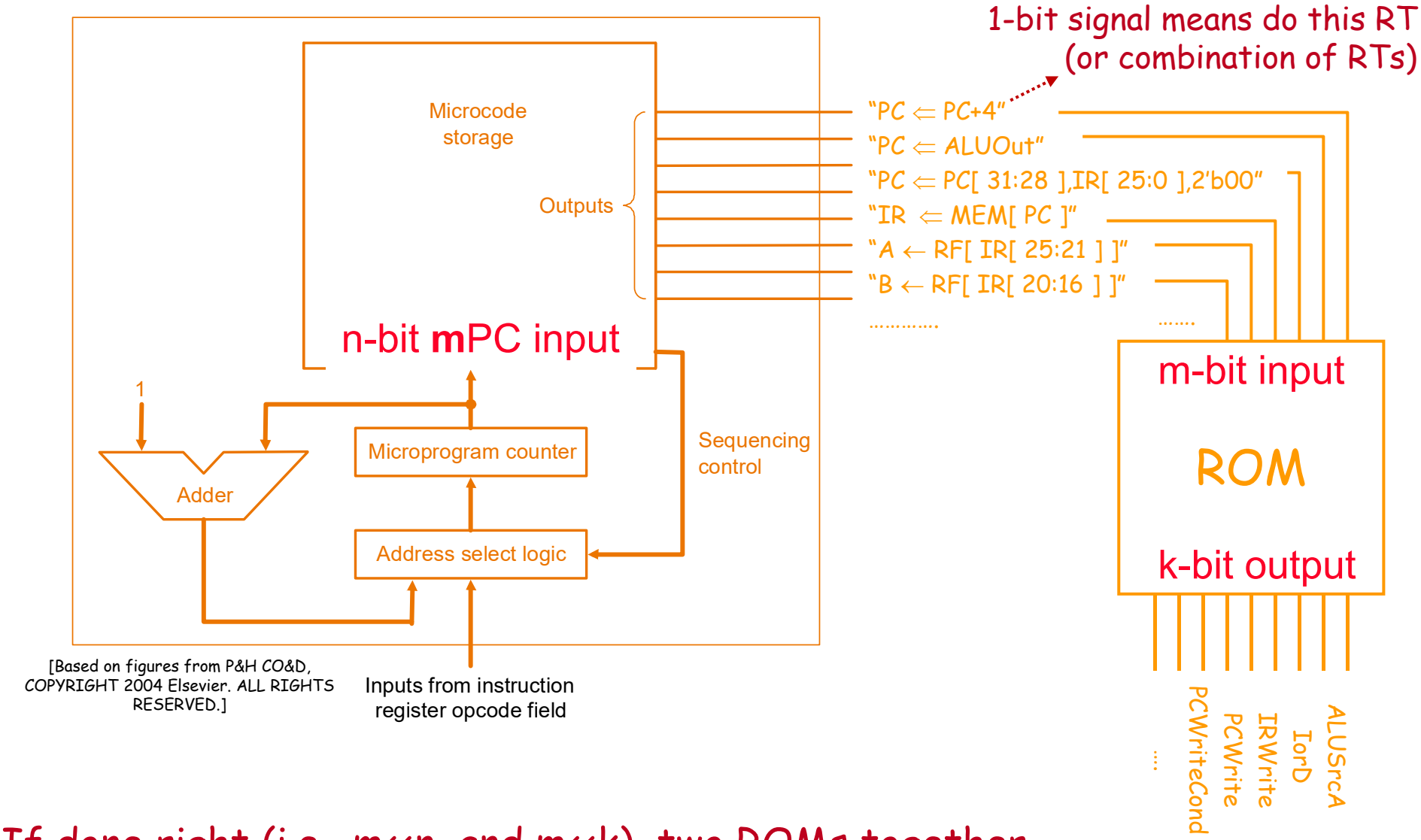
RTs in each state corresponds to some setting of the control signals

Horizontal Microcode



Control Store: $2^n \times$ k bit (not including sequencing)

Vertical Microcode



If done right (i.e., m<<n, and m<<k), two ROMs together ($2^n \times m + 2^m \times k$ bit) should be smaller than horizontal microcode ROM ($2^n \times k$ bit)

Microcoding for CISC

- Can we extend the μcontroller and datapath ?
  - To support a new instruction I haven't thought of yet
  - To support a complex instruction, e.g. polyf
- Yes, and probably more
  - If I can sequence an arbitrary RISC instruction, then I can sequence an arbitrary "RISC program" as a μprogram sequence
  - Will need some μISA state (e.g. loop counters) for more elaborate μprograms
  - More elaborate μISA features also make life easier
- μcoding allows very simple datapath to do very powerful computation
  - A datapath as simple as a Turning machine is universal
  - μcode enables a minimal datapath to emulate any ISA you like (with a very large slow down)

Nanocode and Millicode

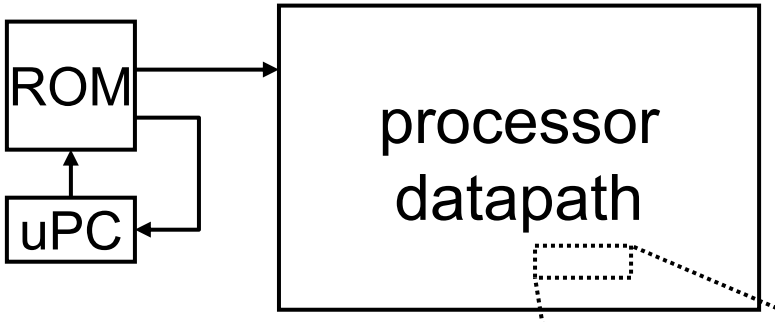
- Nanocode
  - Another level **below** μcode
  - μprogrammed control for sub-systems (e.g., a complicated floating-point module) that acts as a slave in a μcontrolled datapath
  - e.g., The polyf sequence may be generated by a separate nanocontroller in the FPU → more-fine grained coding
- Millicode
  - A level **above** μcode
  - ISA-level subroutines hardcoded into a ROM that can be called by the μcontroller to handle really complicated operations

    e.g., To add polyf to MIPS ISA, one may code up polyf as a software routine that is called by the μcontroller when the polyf **opcode** is decoded → "Instruction emulation"
- In both cases, we don't complicate the μcontroller for polyf support

The power of abstractions!!

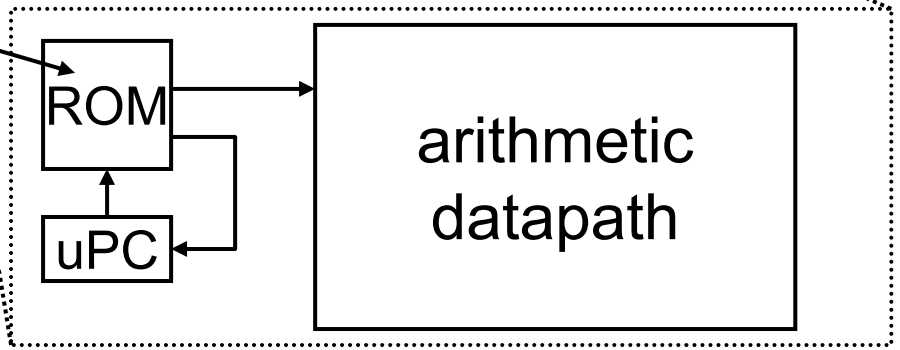
Nanocode Example

a "**m**coded" processor implementation



We refer to this as "nanocode" when a **m**coded subsystem is embedded in a **m**coded system

a "**m**coded" FPU implementation



Now you know how to design a microcontrol-based multi-cycle CPU.

Question?

Announcements: *Homework #2 will be posted soon.*

Reading: *Finish reading P&H Ch.4*

Handouts: *None*