



Computer Architecture

Lecturer: Mihai Negru

2nd Year, Computer Science

Lecture 7: Multi-Cycle CPU Design (2)

Control Unit Design

<http://users.utcluj.ro/~negrum/>



Multi-Cycle Processor Design



- Step-by-step Processor Design → Multi cycle MIPS
 - Step 1: ISA → Abstract RTL
 - Step 2: Components of the Data-Path
 - Step 3: RTL + Components → Data-Path
 - Step 4: Data-Path + Abstract RTL → Concrete RTL
 - **Step 5: Concrete RTL → Control**
- Mux-based multi-cycle data-path designed in the previous lecture



Multi-Cycle CPU Design – Summary



- Five Execution Phases
 - Instruction Fetch
 - Instruction Decode and Register Fetch
 - Execution, Memory Address Computation, or Branch / Jump Completion
 - Memory Access or Arithmetical – Logical instruction completion
 - Write-back
- Instructions take from 3 to 5 clock cycles
- In one clock cycle all operations are done in parallel, not sequential!
 - $T_0 \rightarrow IR \leftarrow M[PC]$ and $PC \leftarrow PC+4$ are done simultaneously
- Between Clock T1 and Clock T2 the control unit will select the next step in accordance to the instruction type



Multi-Cycle CPU Design – Control Signals



T	lorD	Mem Read	Mem Write	IR Write	Reg Dst	Mem toReg	Reg Write	Ext Op	ALU SrcA	ALU SrcB	ALU Op	PC Src	PC WrCd	PC Wr	
T0	0	1	0	1	x	x	0	x	0	1	add	0	0	1	IF
T1	x	0	0	0	x	x	0	1	1	3	add	x	0	0	ID
T2	x	0	0	0	x	x	0	x	1	0	fun	x	0	0	Ex R-T
T3	x	0	0	0	1	0	1	x	x	x	x	x	0	0	Wb R-T
T2	x	0	0	0	x	x	0	0	1	2	or	x	0	0	Ex ORI
T3	x	0	0	0	0	0	1	x	x	x	x	x	0	0	Wb ORI
T2	x	0	0	0	x	x	0	1	1	2	add	x	0	0	Ex LW
T3	1	1	0	0	x	x	0	x	x	x	x	x	0	0	M LW
T4	x	0	0	0	0	1	1	x	x	x	x	x	0	0	Wb LW
T2	x	0	0	0	x	x	0	1	1	2	add	x	0	0	Ex SW
T3	1	0	1	0	x	x	0	x	x	x	x	x	0	0	M SW
T2	x	0	0	0	x	x	0	x	x	x	sub	1	1	0	Ex BEQ
T2	x	0	0	0	x	x	0	x	x	x	x	2	0	1	Ex J

Table 1: The Values of the Control Signals in each Clock Cycle

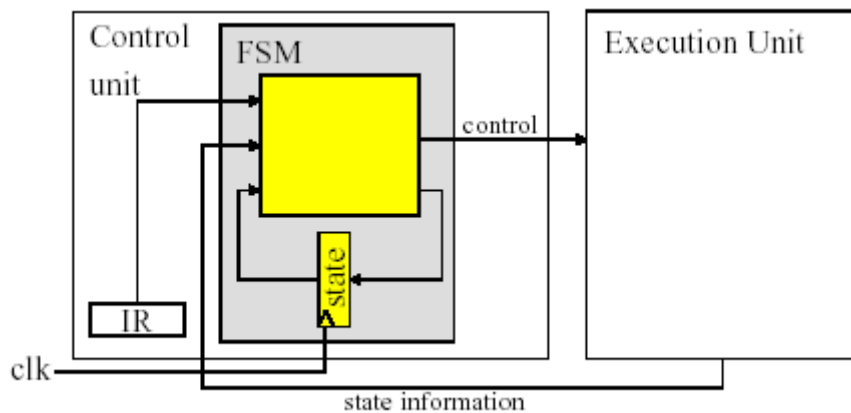
- Execution phases: IF, ID, Ex – Execute, M – Memory, Wb – Write back
- Instructions: R – R-type, LW – Load, SW – Store, BEQ – Branch , J – Jump , ORI – I-type
- ExtOp: 1/0 → 1 – arithmetic, 0 – logical operations



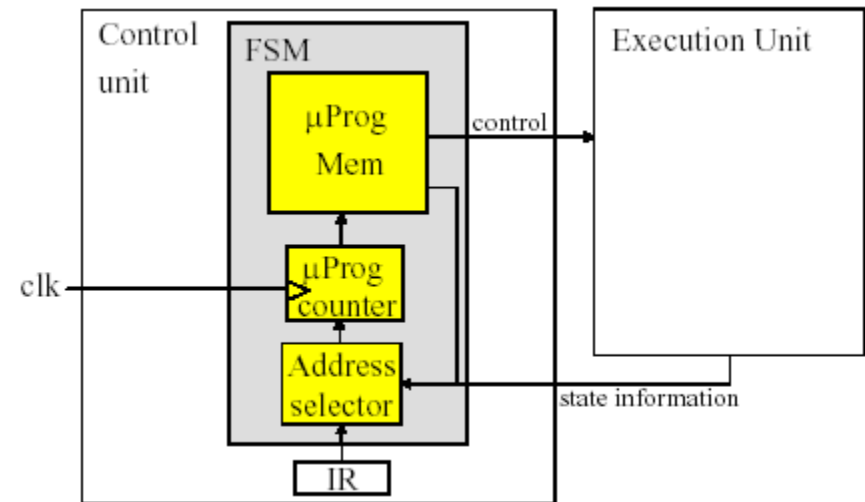
Multi-Cycle CPU Design – Step 5



- Control Unit implementation
 - Hardwired
 - Micro-Programmed
- Hardwired vs. Micro-Programmed
 - Hardwired units are faster
 - Hardwired design is complex if large
 - Bugs in hardwired design cannot be fixed in field
 - Emulation is easy with micro-coding



Hardwired Control Unit



Micro-Programmed Control Unit



Multi-Cycle CPU Design – Step 5



- Control Unit implementation can be derived from specification
 - Hardwired
 - Micro-Programming
- Hardwired control unit
 - We can specify the Control Unit as a finite state machine
 - We'll use a Moore machine (output based only on current state)
 - Value of control signals is dependent upon:
 - What instruction is being executed
 - Which step is being performed
 - Each entry in the **Control Signals Values Table** is a state in our FSM

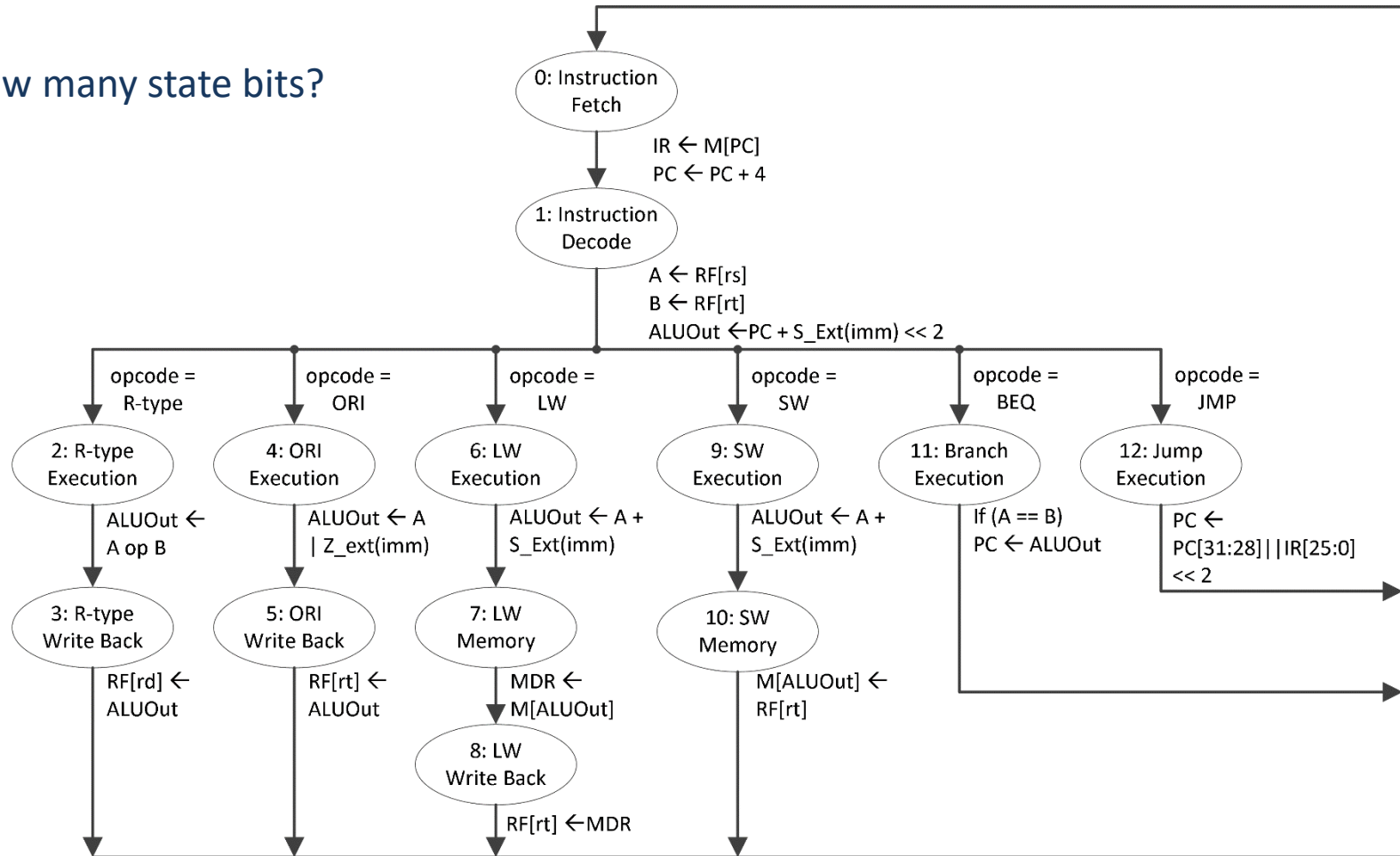


Hardwired Control Unit – FSM



- Finite State Machine for MIPS-lite Multi-Cycle CPU: FSM 1
 - FSM1 is based on the Control Signals Values table (table 1)

How many state bits?



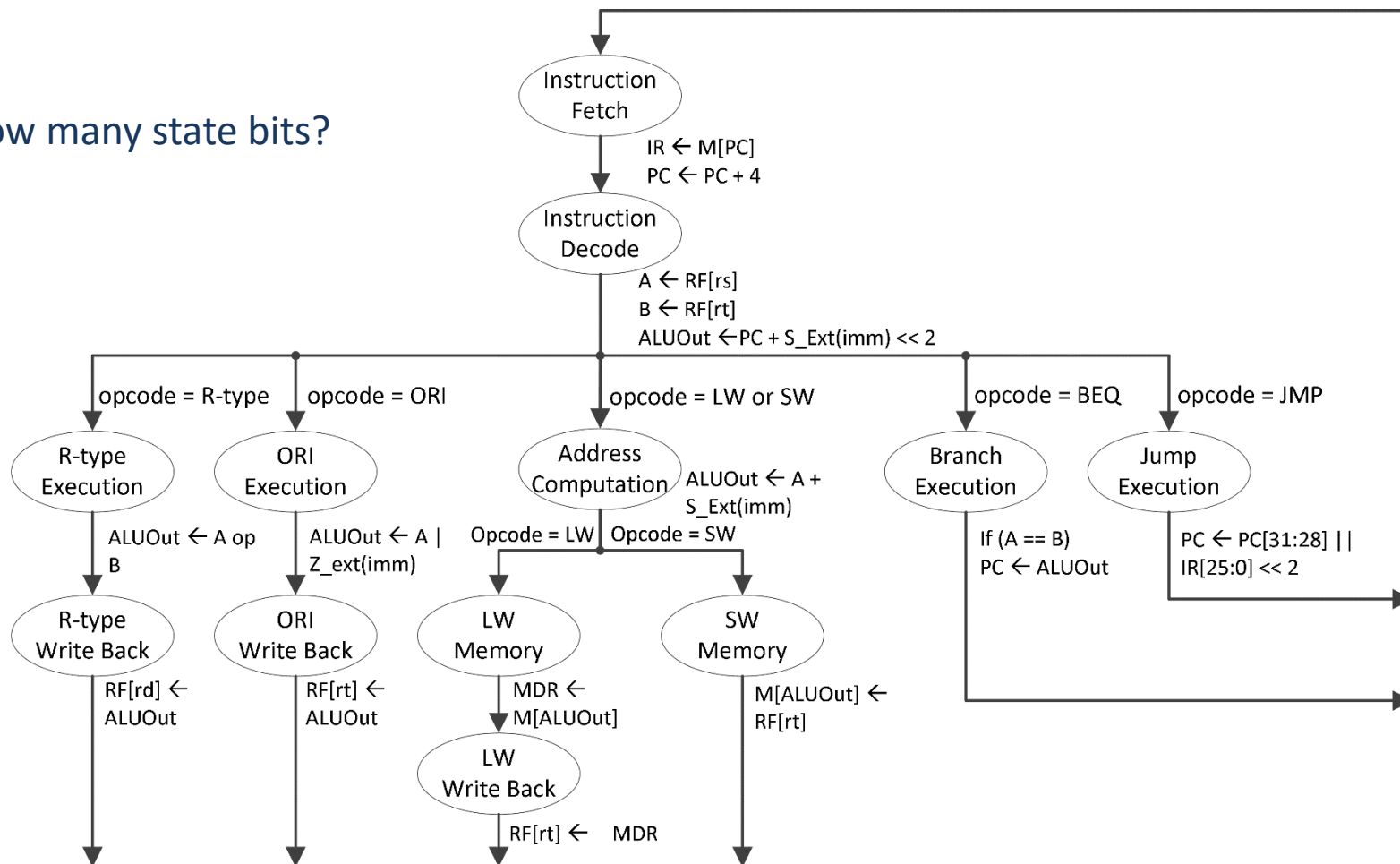


Hardwired Control Unit – FSM



- Optimized FSM1 – unify Execution for LW and SW: FSM 2

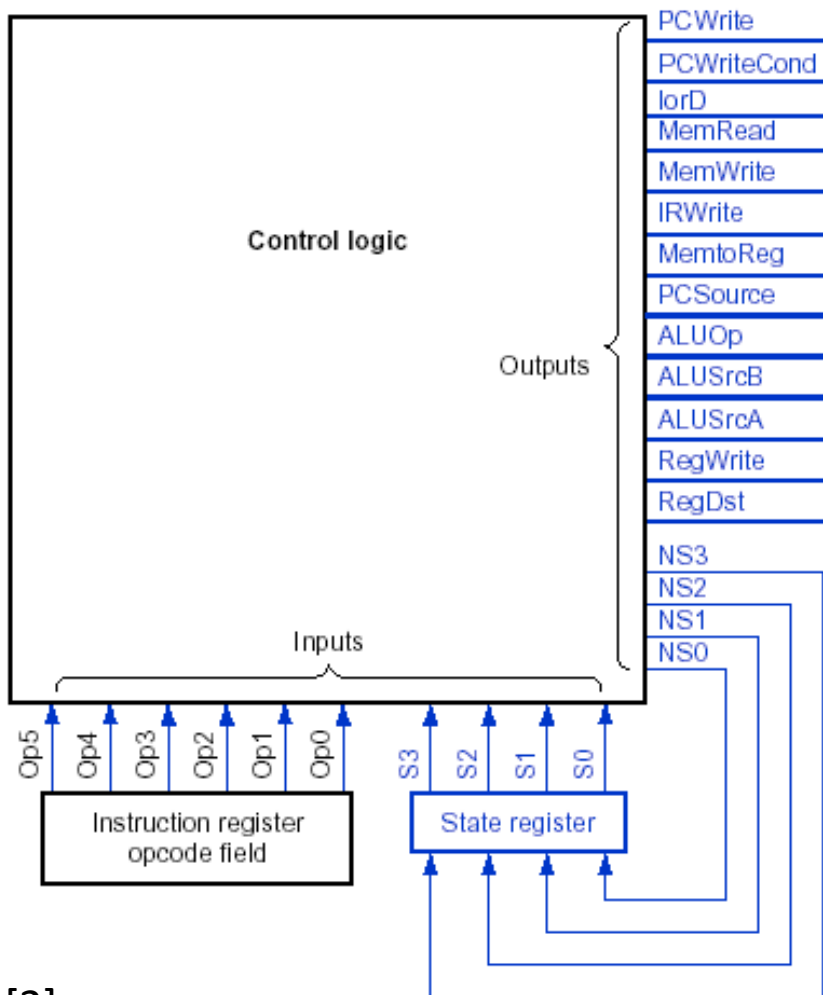
How many state bits?





Hardwired Control Unit

- Hardwire implementation for MIPS Multi-Cycle Control Unit



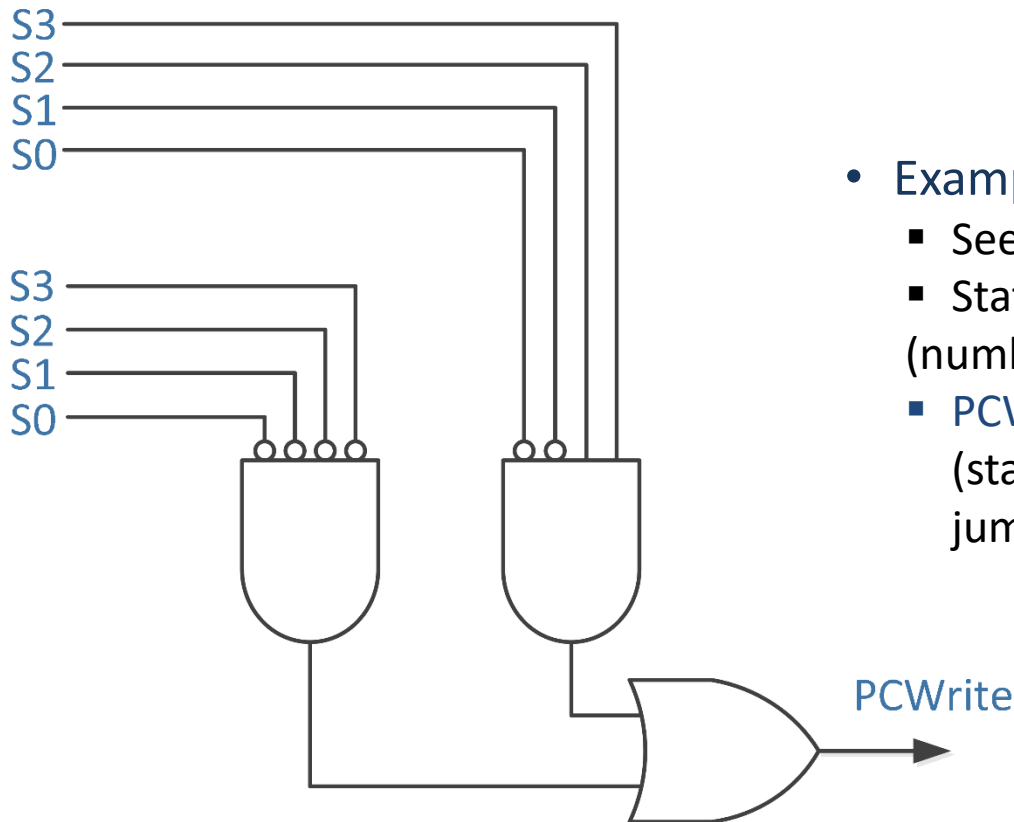
[2]

- Control Unit
 - State register to hold the state
 - Control logic
 - assigns the values for the control signals in each state
 - Current state + opcode field decode the next state
- Possible implementations:
 - Programmable Logic Array (PLA) to implement the control logic
 - One Flip-Flop per state to implement the FSM
 - Sequence/Jump Counter + Decoder to implement the FSM
 - ...



Hardwired Control Unit

- PLA (programmable logic array) based control logic
 - A net of **AND** gates; the outputs are connected by **OR** gates
 - Every control signal is set by the current state
 - **Next state** derived from **current state** and the **opcode** field



- Example – PCWrite :
 - See table 1, FSM1.
 - States encoded on 4 bits (numbering top → down, left → right)
 - **PCWrite** must be set in Instruction Fetch (state 0 = “0000”) and when executing a jump (state 12 = “1100”)



Hardwired Control Unit – 1 FF/state

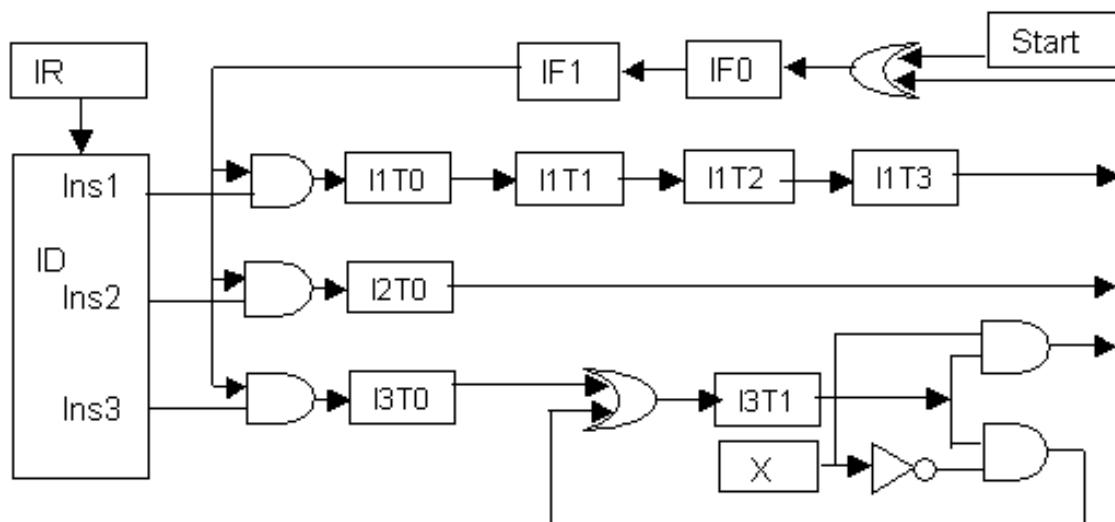


- State encoding Types

- Compact binary encoded FSM

- The state numbers depend on instruction decode and execution phases
 - → Hard to extend or modify

- One flip-flop per state (One-Hot encoding)

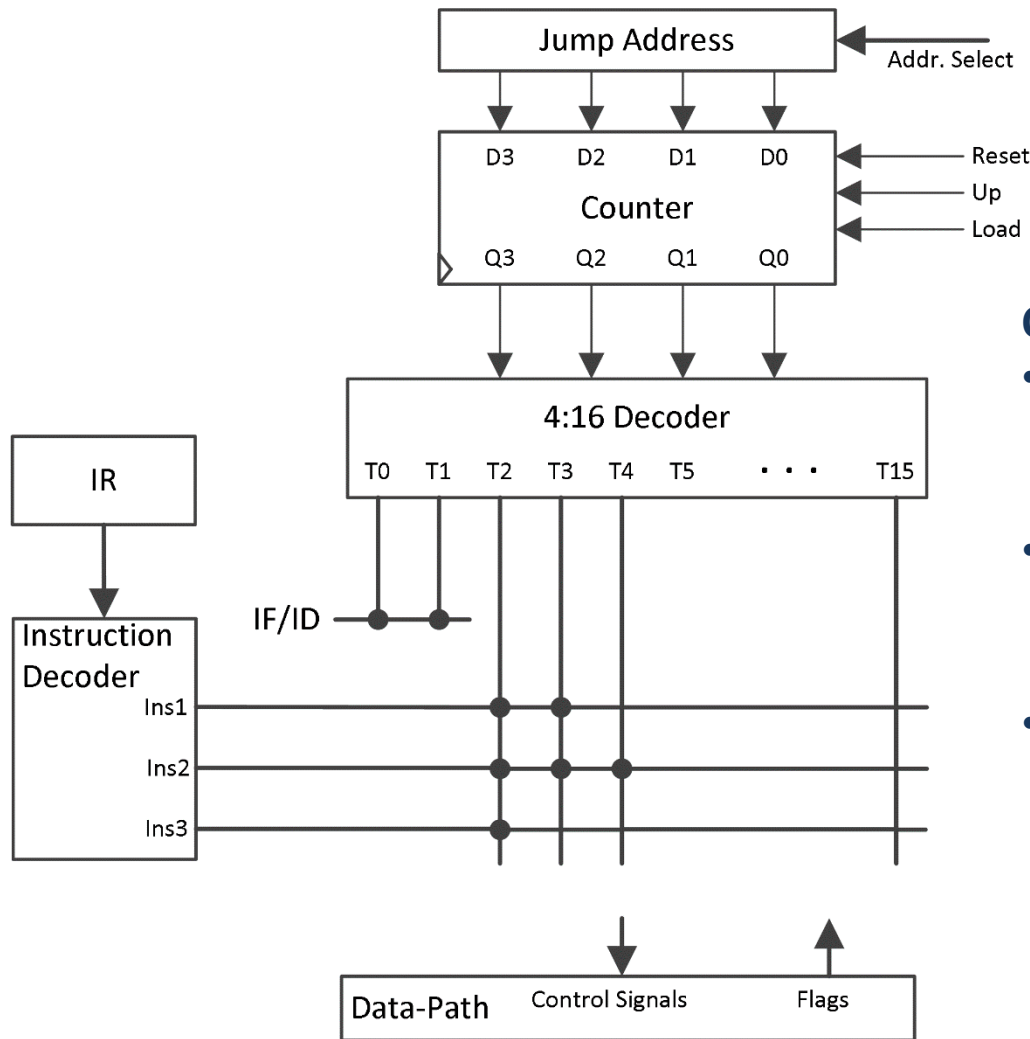


One Flip-Flop per State Control Unit

- Start: Synchronized, a single clock period duration signal
- X: Condition Flag
- IF0, IF1: Instruction Fetch and decode execution phases
- IjTi: Instruction execution phases
- The instruction decode and execution phases are independent
- → Easy to extend or modify



Hardwired Control Unit – Sequence Counter



Sequence/Jump Counter Control Unit

- IF, ID: T0 and T1
- After IF, InsX & T2 determines the start of a new instruction execution
- After completion → Reset → T0 and IF for the new instruction

Counter Operation

- **Reset:** Counter Reset
 - selects the T0 column through the Decoder
- **Up:** Count Up validation
 - Implicitly asserted
 - Load and Reset have priority
- **Load:** Counter Load for micro-jumps inside the counter
 - The micro-jump addresses are selected by the currently active control signals from the control matrix

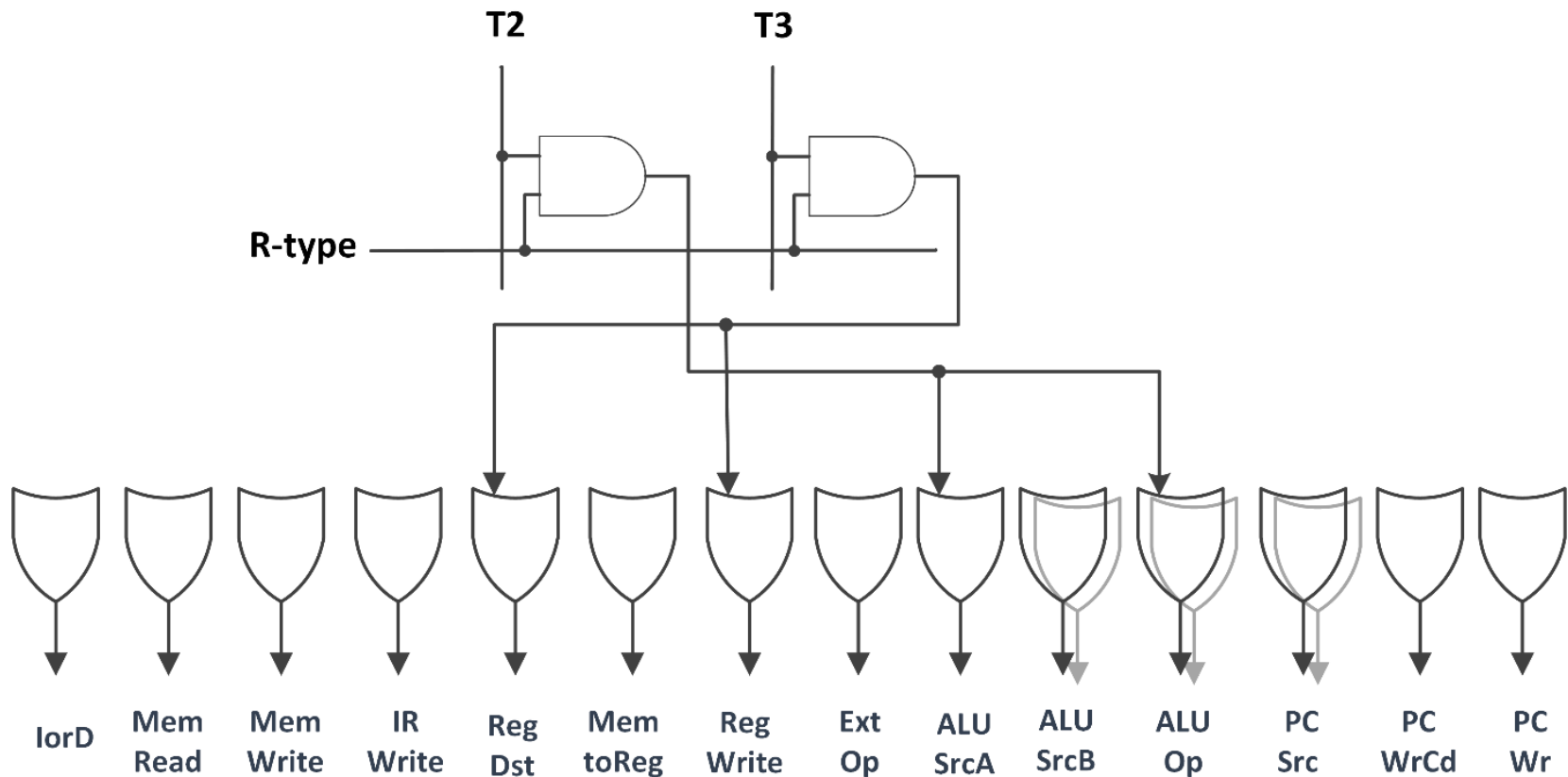
Reduction of states possible – depends on implementation



Hardwired Control Unit – Sequence Counter



- Sequence/Jump Counter Control Unit
 - Example for R-type instructions
 - Sequencing: T2 – implicit Up, T3 – Reset (jump to IF – T0)



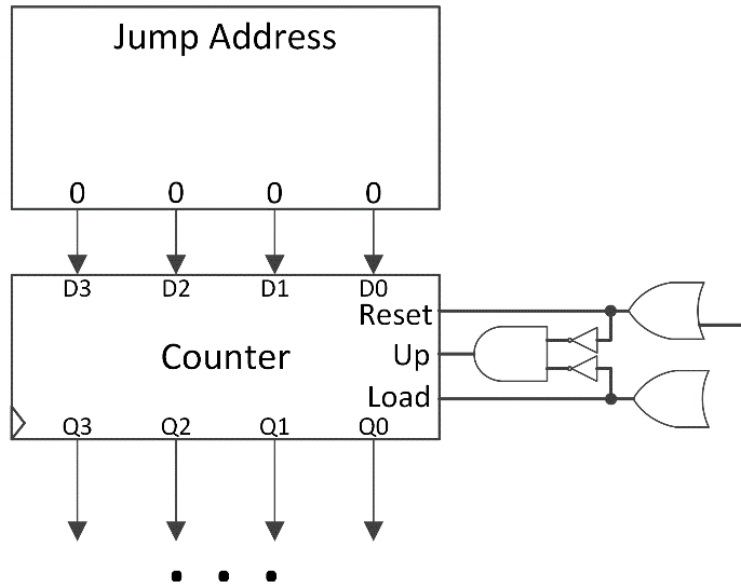
Hardwired Control Unit Decode Logic for R-type instruction



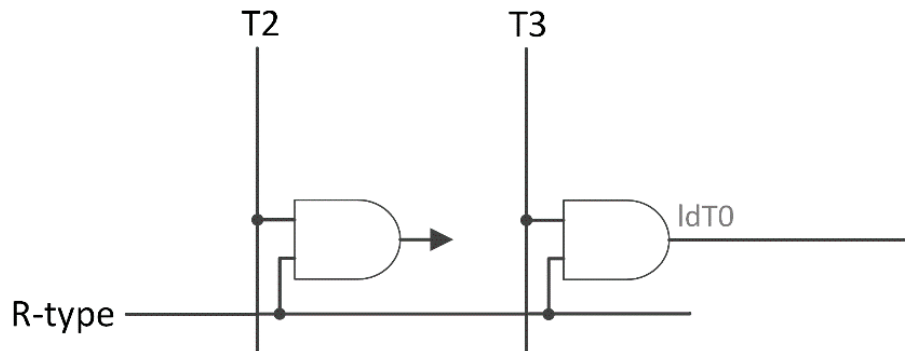
Hardwired Control Unit – Sequence Counter



- Sequencing logic example: R-type Instructions



- T2 – implicit Up
- T3 – Reset (jump to IF – T0)

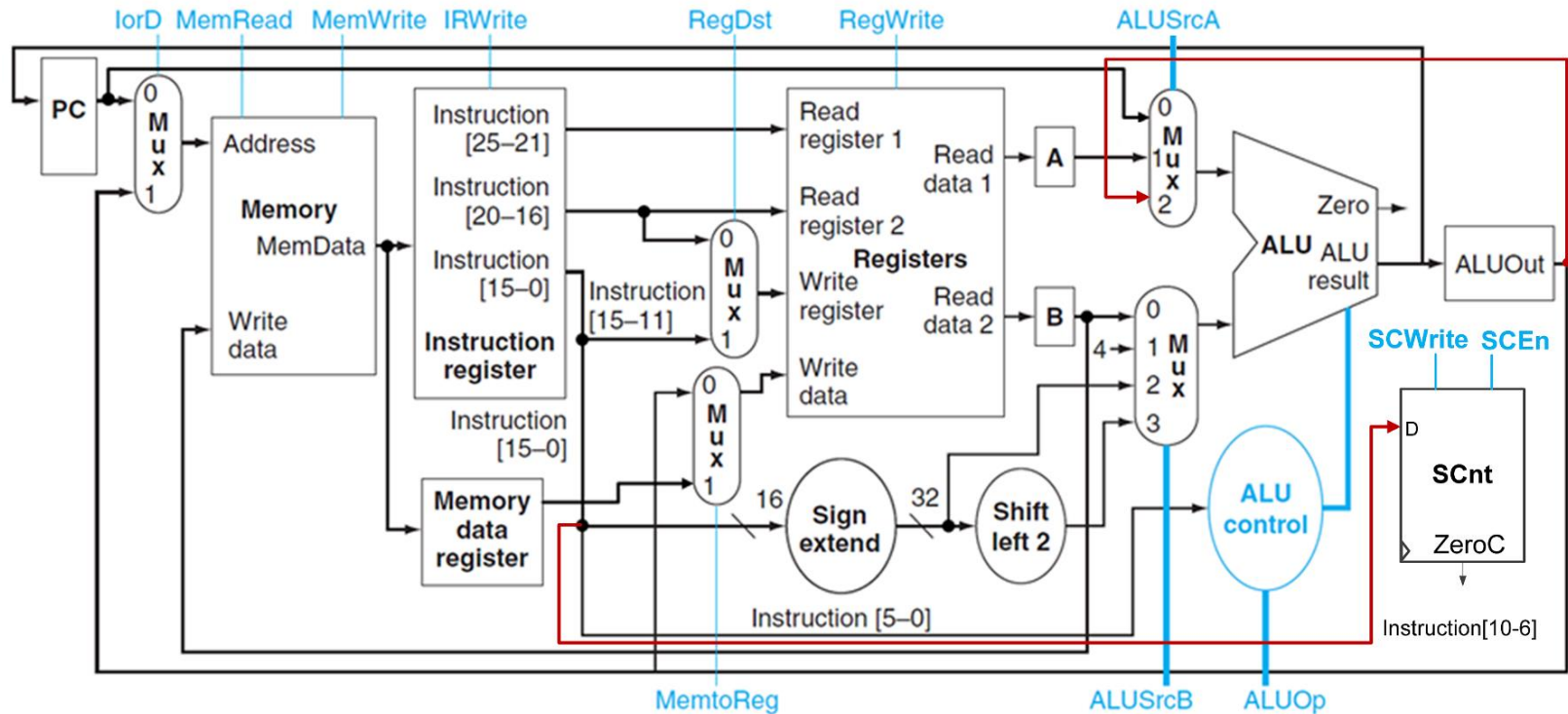




- Example – extend for instructions that take a variable number of clock cycles
- **R-type: sll \$rd, \$rs, sa** RTL Abstract: $RF[rd] \leftarrow RF[rs] \ll sa$
 - Suppose ALU performs only 1-bit shifts
 - We must extend the data-path in order to allow
 - A mechanism of counting the clock cycles: dedicated counter **SCnt** with parallel load (sa is loaded when **SCWrite** is asserted), count down enable (**SCen**), 1-bit zero detector signal (**ZeroC**)
 - A connection from **ALUOut** to one of the **ALU inputs**, in order to repeat the shifting process (as long as it is necessary)
 - The control unit will treat this R-type instruction separately (dedicated line from Instruction Decoder, to the sequence counter)



Hardwired Control Unit – Sequence Counter

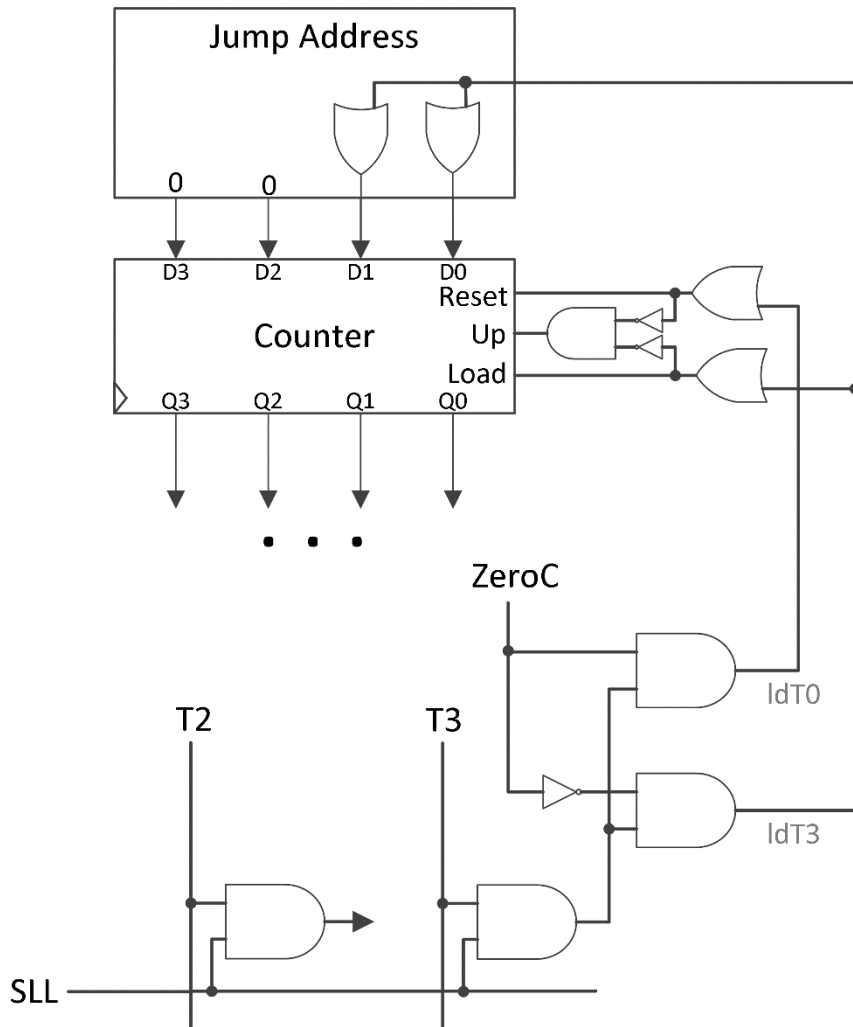


$T0 \rightarrow$	$IR \leftarrow M[PC], PC \leftarrow PC + 4;$
$T1 \rightarrow$	$A \leftarrow RF[rs], B \leftarrow RF[rt]; SCnt \leftarrow sa$
$SLL \ \& \ T2 \ \& \ ZeroC = 0 \rightarrow$	$ALUOut \leftarrow A \ll 1; SCnt \leftarrow SCnt - 1$
$SLL \ \& \ T2 \ \& \ ZeroC = 1 \rightarrow$	$ALUOut \leftarrow A;$
$SLL \ \& \ T3 \ \& \ ZeroC = 0 \rightarrow$	$ALUOut \leftarrow ALUOut \ll 1; SCnt \leftarrow SCnt - 1; \text{repeat (load } T3)$
$SLL \ \& \ T3 \ \& \ ZeroC = 1 \rightarrow$	$R[rd] \leftarrow ALUOut; \text{load } T0$

What are the values of the control signals in each state?



Hardwired Control Unit – Sequence Counter



T2: Implicit Up

T3: (ZSC = 0) \rightarrow IdT3; Repeat the same state
(ZSC = 1) \rightarrow IdT0; Reset (go to IF – T0)

Shift Left Logical: SLL – sequencing



Multi-Cycle CPU Design – Step 5



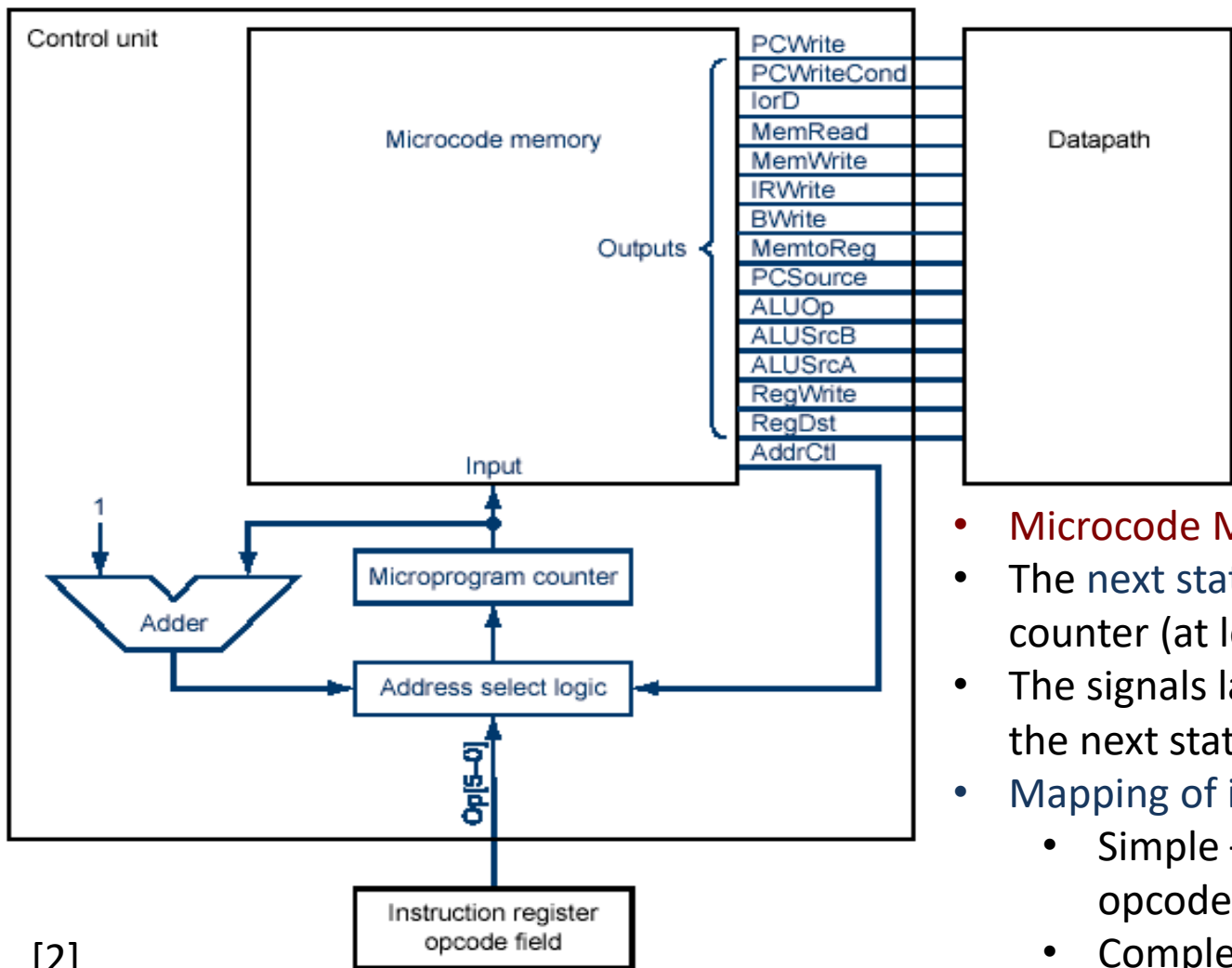
- Disadvantages of Using Hardwire Control Units
 - The complexity of the FSM depends on Data-Path complexity
 - Real processors are complex: over 100 instructions with 1 to 20 cycles.
 - → Use micro-programming
- Micro-Programmed Control Unit
 - Appropriate if hundreds of opcodes, modes, cycles, etc.
 - Control signals are specified symbolically using **μinstructions**
 - Possible **Sequencing Capabilities** in a Micro-Programmed Control Unit
 - Incrementing of the control address register
 - Unconditional and conditional branches
 - Mapping from the machine instruction's operation code to the address of the corresponding μinstruction sequence in control memory
 - A facility for subroutine call and return



Micro-Programmed Control Unit



• Micro-Programmed Control Unit Block Diagram



- **Microcode Memory = Control Memory**
- The next state is computed using a counter (at least in some states)
- The signals labeled **AddrCtl** control how the next state is determined → MUX
- Mapping of instructions to μ operations
 - Simple – concatenation between opcode and address
 - Complex – ROM or PLA

[2]



- Variations on Micro-Programming
 - “Horizontal” micro-code → Example on next slides
 - Each data-path control signal is represented by a bit in the μ instruction
 - Horizontal μ code has wider μ instructions
 - Multiple parallel operations per μ instruction
 - Fewer steps per instruction
 - Simple encoding → more bits
 - “Vertical” micro-code
 - Encode fields to save ROM space
 - Vertical μ code has narrower μ instructions
 - Typically a single data-path operation per μ instruction
 - Separate μ instructions for branches
 - More steps per instruction
 - More compact → less bits
 - Nano-coding
 - Two level microprogramming
 - Tries to combine best of horizontal and vertical μ -code



- Micro-Programming Pros and Cons
 - Ease of design
 - Flexibility
 - Easy to adapt to changes in organization, timing, technology
 - Can make changes late in design cycle, or even in the field
 - Can implement very powerful instruction sets (just more control memory)
 - Generality
 - Can implement multiple instruction sets on same machine
 - Can tailor instruction set to application requirements
 - Compatibility
 - Many organizations, same instruction set
 - Slow (ROM is no longer faster than RAM)



Micro-Programmed Control Unit



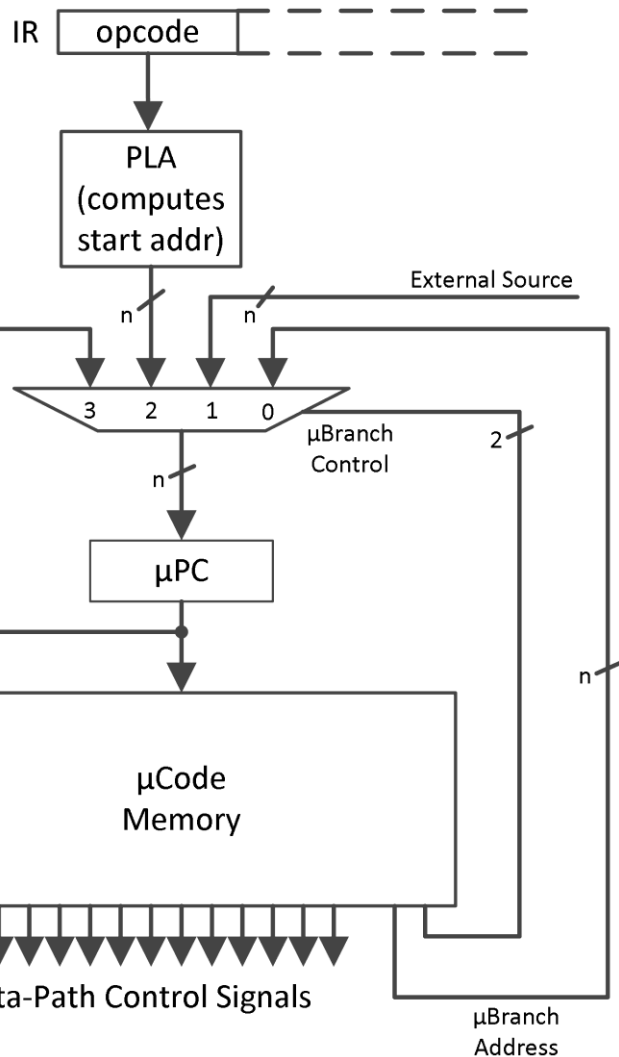
- Designing a μ instruction set
 1. Start with list of control signals
 2. Group signals together that make sense (vs. random): called **fields**
 3. Places fields in some logical order (e.g., ALU operation & ALU operands first and μ instruction sequencing last)
 4. Create a symbolic legend for the μ instruction format, showing name of field values and how they set the control signals
 5. To minimize the width, encode in the same field operations that will not be used at the same time (vertical micro-programming)



Micro-Programmed Control Unit – V1



• Micro-Programmed Control Unit Example for FSM1



μInstruction fields

Data-Path Control Signals	μBranch Address	μBranch Control
---------------------------	-----------------	-----------------

- The data-path control signals are defined in table 1
- The μBranch address: 4 bits (13 states)
- The μBranch control: select signal for the multiplexer – 2 bits
- μBranch control
 - 00 → μBranch address
 - 01 → External Source (not used in this example)
 - 10 → The first μinstruction address for the mapped operation code
 - 11 → Next sequential μinstruction address



Micro-Programmed Control Unit – V1



Addr.	lorD	MemRead	MemWrite	IRWrite	RegDst	Mem to Reg	Reg Write	ExtOp	ALUSrcA	ALUSrcB	ALUOp	PCSrc	PCWrCd	PCWr	μBranch Address	μBranch Control	Execution Phase
No.	1b	1b	1b	1b	1b	1b	1b	1b	1b	2b	2b	2b	1b	1b	4b	2b	
00	0	1	0	1	x	x	0	x	0	1	add	0	0	1	x	3	IF
01	x	0	0	0	x	x	0	1	1	3	add	x	0	0	x	2	ID
02	x	0	0	0	x	x	0	x	1	0	fun	x	0	0	x	3	Ex R-T
03	x	0	0	0	1	0	1	x	x	x	x	x	0	0	0000	0	Wb R-T
04	x	0	0	0	x	x	0	0	1	2	or	x	0	0	x	3	Ex ORI
05	x	0	0	0	0	0	1	x	x	x	x	x	0	0	0000	0	Wb ORI
06	x	0	0	0	x	x	0	1	1	2	add	x	0	0	x	3	Ex LW
07	1	1	0	0	x	x	0	x	x	x	x	x	0	0	x	3	M LW
08	x	0	0	0	0	1	1	x	x	x	x	x	0	0	0000	0	Wb LW
09	x	0	0	0	x	x	0	1	1	2	add	x	0	0	x	3	Ex SW
10	1	0	1	0	x	x	0	x	x	x	x	x	0	0	0000	0	M SW
11	X	0	0	0	x	x	0	x	x	x	sub	1	1	0	0000	0	Ex Br
12	x	0	0	0	x	x	0	x	x	x	x	2	0	1	0000	0	Ex J

The μCode Memory for the selected instructions

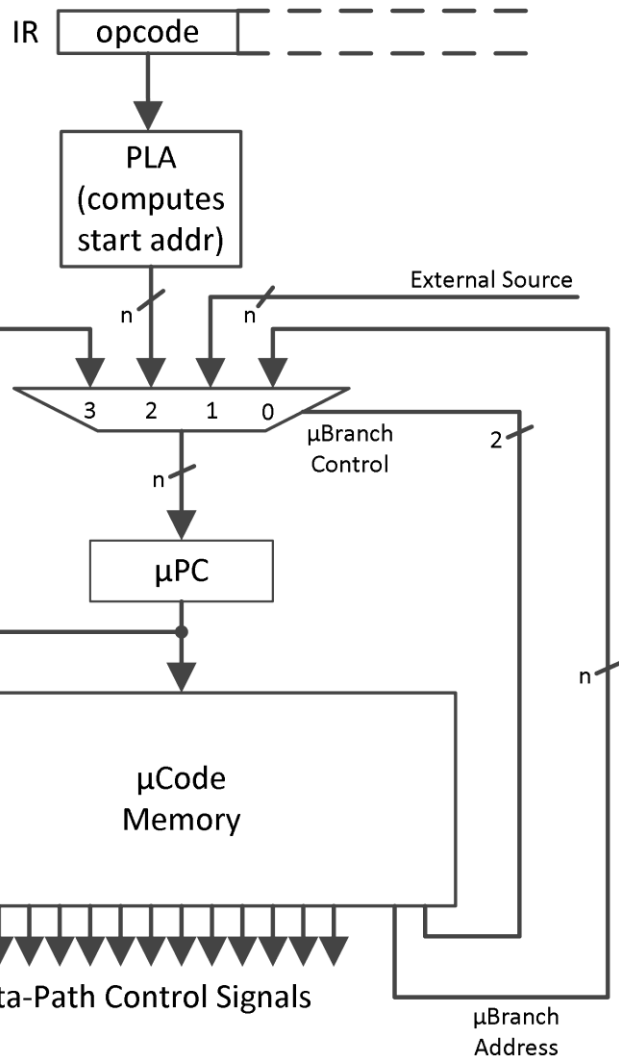
- Horizontal micro instruction length: **23 bits**
- The addresses correspond to the order in table 1
- For the selected instructions the μBranch field is used only for IF μAddress generation
- If we connect to the External Source the IF μAddress → shorter μInstruction length



Micro-Programmed Control Unit – V1



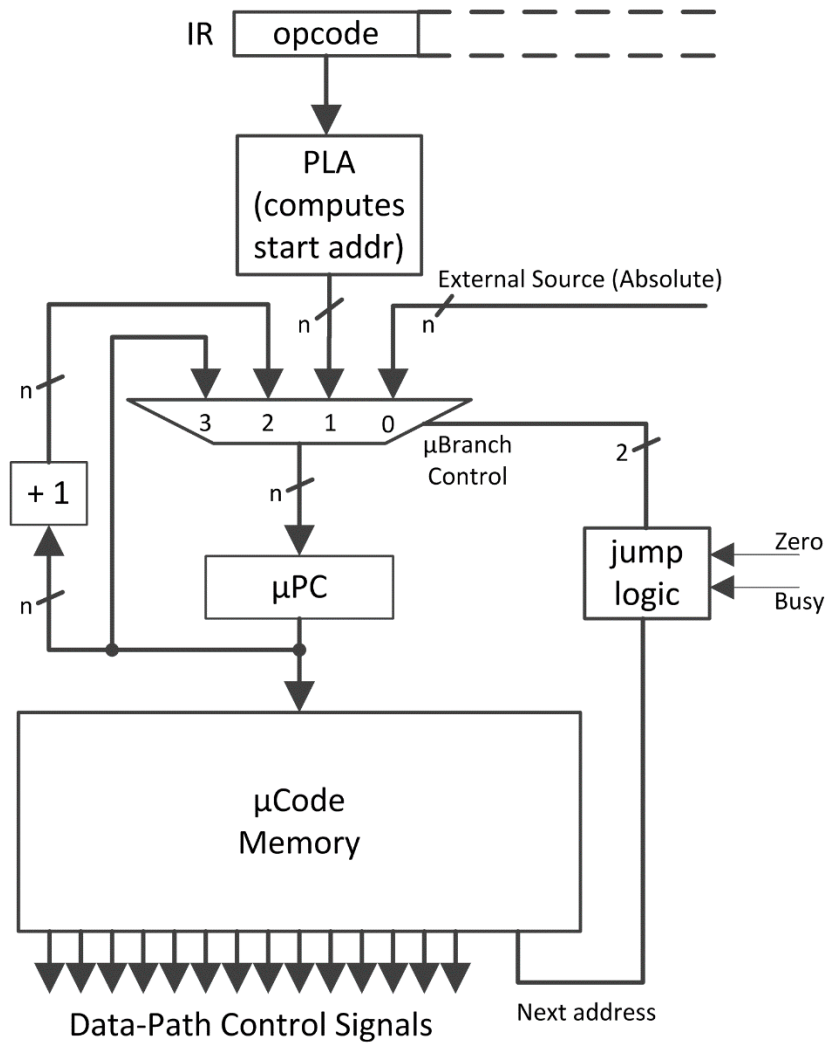
• Micro-Programmed Control Unit Example for FSM1



- μ Branch control
 - 00 \rightarrow μ Branch address
 - 01 \rightarrow External Source (not used in this example)
 - 10 \rightarrow The first μ instruction address for the mapped operation code
 - 11 \rightarrow Next sequential μ instruction address
- **Disadvantage:** the select logic does not depend on state bits \Rightarrow one cannot implement more complex instructions (SLL from our previous example)
- **Possible variation** – different inputs to the sequencing Multiplexer and conditional logic for the select signal of the multiplexer



Micro-Programmed Control Unit – V2



μ-Programmed Control Unit, V2

Differences from V1:

- μPC and $\mu\text{PC}+1$ inputs
- The μBranch address and μBranch control replaced by Next address
- External – absolute address of IF from μCode Memory
- Sequencing based on Jump Logic – considering the data-path status signals and Next address

μControl values

- next $\rightarrow \mu\text{PC}+1$
- spin \rightarrow if (busy) then μPC else $\mu\text{PC}+1$
- fetch \rightarrow absolute
- dispatch $\rightarrow \text{PLA} = \text{MAP}(\text{opcode})$ or a decoder
- feqz \rightarrow if (zero) then absolute else $\mu\text{PC}+1$ (fetch if equal zero)
- fnez \rightarrow if (zero) then $\mu\text{PC}+1$ else absolute (fetch if not equal zero)



- Micro-Programmed Control Unit Possible Types
 - V1: The Sequencer's MUX inputs: $\mu\text{PC}+1$, MAP (Opcode), External Source, μBranch Address. Unconditional MUX Selection Logic from the $\mu\text{instructions}$.
 - V2: The Sequencer's MUX inputs: MAP (opcode), External Source (absolute), $\mu\text{PC}+1$, μPC . Conditional MUX Selection Logic: Condition selection code from the $\mu\text{instructions}$ and external condition flag evaluation logic.
 - V3: The Sequencer's MUX inputs: $\mu\text{PC}+1$, MAP1 (opcode), MAP2 (Opcode), Instruction Fetch $\mu\text{Address}$. Unconditional MUX Selection Logic from the $\mu\text{instructions}$ (Vertical Micro-Programming).
 - V4: The Sequencer's MUX inputs: $\mu\text{PC}+1$, MAP(Opcode), External Source, μBranch Address. Conditional MUX Selection Logic: Condition selection code from the $\mu\text{instructions}$ and external condition flag evaluation logic.
 -



Multi-Cycle CPU Design – Exceptions



- Definition: Event causes unexpected transfer of control
- Types:
 - Exceptions[overflow] → generated inside the processor
 - Interrupts [I/O] → associated with external events
 - MIPS Exceptions: undefined instruction or arithmetic overflows
 - Exception Detection – How to discover exception?
 - Exception Handling – What to do?
- Exception Detection:
 - Undefined Instruction
 - Add an `IllegalOp` State to the FSM
 - Every unknown instruction (undefined opcode) → transitions to the `Exception State`
 - Arithmetic Overflow
 - ALU has overflow detection logic → create a new `Overflow State` to handle overflow



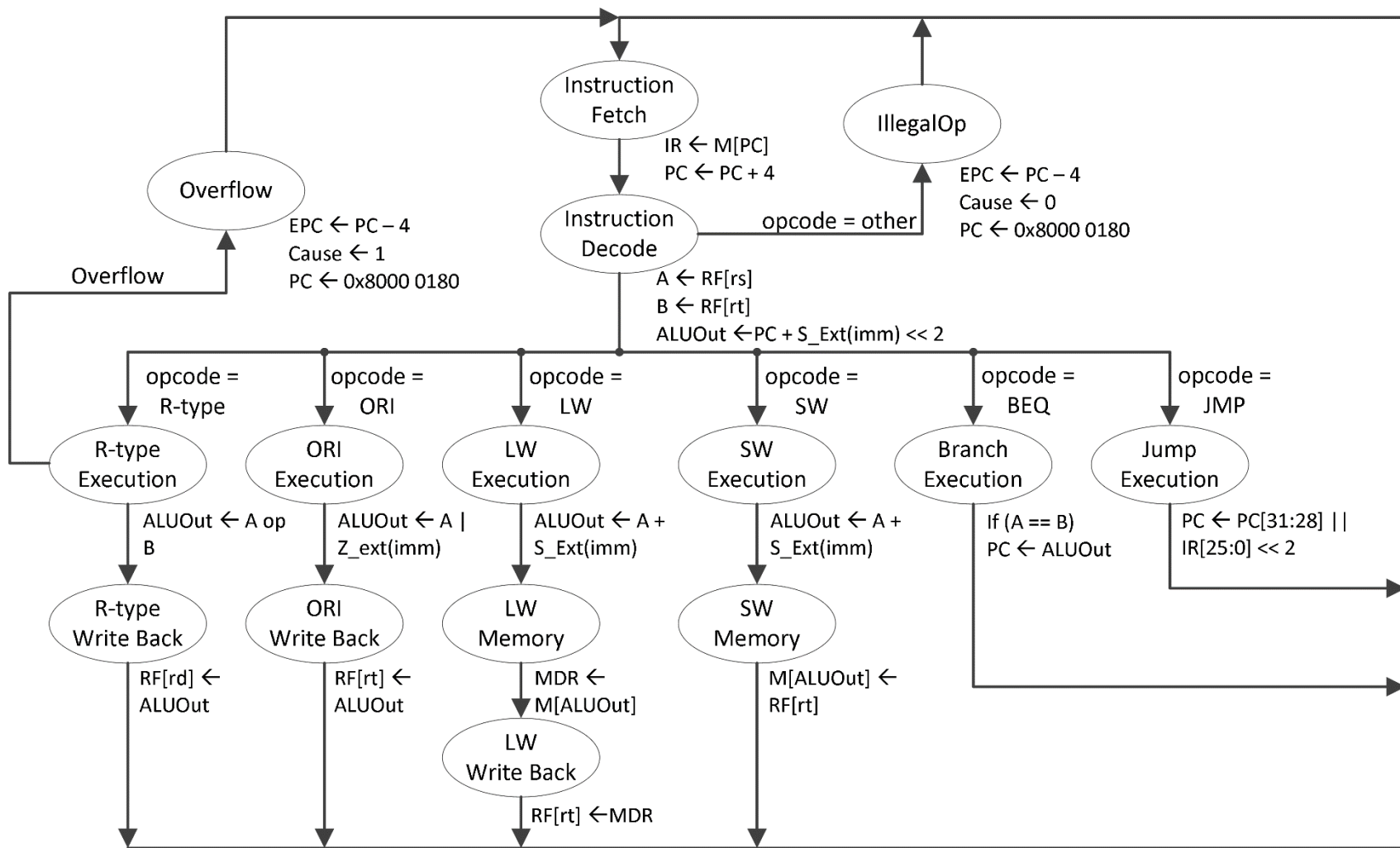
- Exception handling
 - Two Techniques: EPC / Cause Registers and Vectored Interrupts
 - Vectored Interrupts
 - Each exception has a distinct address A_E associated with it
 - Exception detected $\rightarrow A_E$ for that exception written to PC
 - EPC / Cause: MIPS
 - Exception Program Counter Register – EPC (32 bits): stores the address of the offending instruction: $EPC \leftarrow (PC + 4 - 4) = PC$ (use ALU for subtract 4)
 - Cause Register – 32 bits: stores the cause of the exception:
 - undefined instruction: $Cause \leftarrow 0$
 - arithmetic overflows: $Cause \leftarrow 1$
 - Exception detected
 - \rightarrow Address of instruction saved in EPC
 - \rightarrow Cause register stores an exception type code
 - Exception handler acts on Cause, tries to restart execution at instruction pointed to by EPC



Multi-Cycle CPU Design – Exceptions



- FSM1 with Exceptions handling states





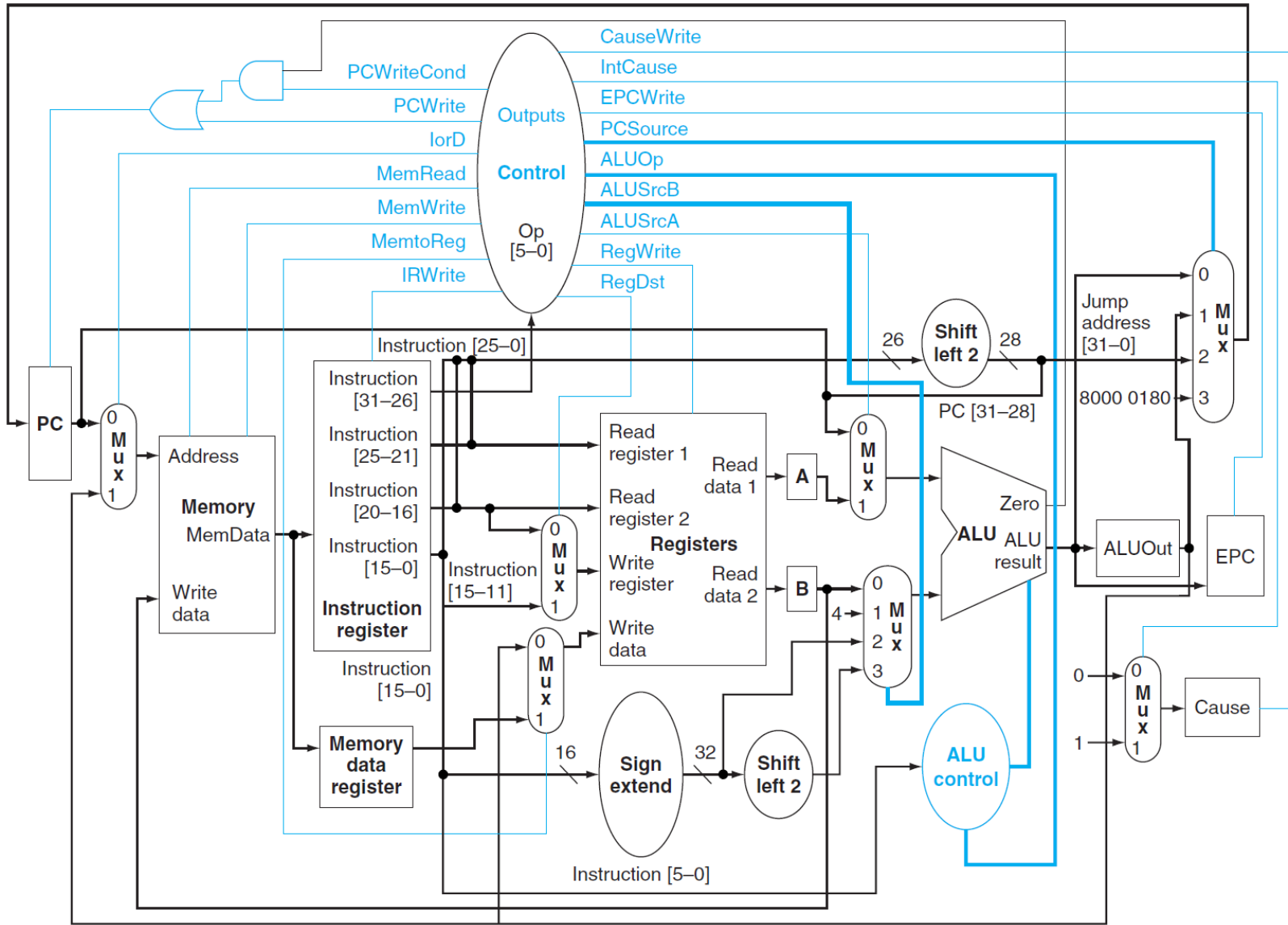
Multi-Cycle CPU Design – Exceptions



- Modifications to MIPS Multi-Cycle Data-Path for Exceptions
 - New Registers – EPC and Cause (32-bit)
 - New Control Signals – EpcWrite, CauseWrite
 - New Control Line: IntCause
 - 0 for undefined instruction
 - 1 for overflow
 - New Mux input for PCsource Signal (3)
 - PC inputs
 - PC + 4
 - Branch Target Address
 - Jump Address
 - Additional input: $A_E = 8000\ 0180_{16}$ in MIPS
- Recall: ALU overflow detection already “installed”



Multi-Cycle CPU Design – Exceptions



[1]



Problems – Homework



- 1-Bus, 2-Bus and 3-Bus based MIPS implementation of the instructions
 - add, sub, and, or, lw, sw, beq, j, addi, andi, ori
 - sll, srl, sra, sllv, srlv, srav
 - slt, slti
 - bne , bgez, bltz,...
 - jr, jal
 -
- Design the control unit in order to reflect the new instructions
 - Hardwired control unit
 - Micro-programmed control unit



References



1. D. A. Patterson, J. L. Hennessy, “Computer Organization and Design: The Hardware/Software Interface”, 3rd edition, ed. Morgan–Kaufmann, 2005.
2. Vincent P. Heuring, Harry F. Jordan, “Computer Systems Design and Architecture”, 2nd Edition
3. CODE3e – Appendix D: Mapping Control to Hardware
4. Microprogramming – *Arvind* Computer Science & Artificial Intelligence Lab M.I.T., Based on the material prepared by Arvind and Krste Asanovic, September 21, 2005
5. MIPS32™ Architecture for Programmers, Volume I: “Introduction to the MIPS32™ Architecture”.
6. MIPS32™ Architecture for Programmers Volume II: “The MIPS32™ Instruction Set”.