Advanced Computer Architecture

Computer Hardware Design

(Single Cycle Datapath and Control Design)

Dr. Salman Afsar

Today's Topics

- Recap: Instruction Set Principles
- Basics of Computer Hardware Design (Review)
- Single Cycle Design
  - Data Path design
  - Control Design
- Summary

Recap: Instruction Set Principles

- Three pillars of Computer Architecture
- Instruction encoding
  - Instruction word length: Fixed, variable and Hybrid length
  - MIPS Instruction word format
- Multimedia and Digital Signal Processor Operands and Operations
- Digital Signal Processing Issues
  - Saturating Add/Subtract
  - Result Rounding
  - Multiply Accumulate

Recap: Instruction Set Principles … Cont'd

- Instruction Set Performance
  - Role of Compiler
  - Impact of Compiler Technology
  - Two ways the interaction of compiler and high-level language affects the use of ISA by a program
    1: How are variables allocated?
    2: How many registers are needed to allocate variables appropriately?
- Three areas of data allocation
  - Local Variable area – Stack
  - Global Data Area
  - Dynamic Object Allocation: Heap

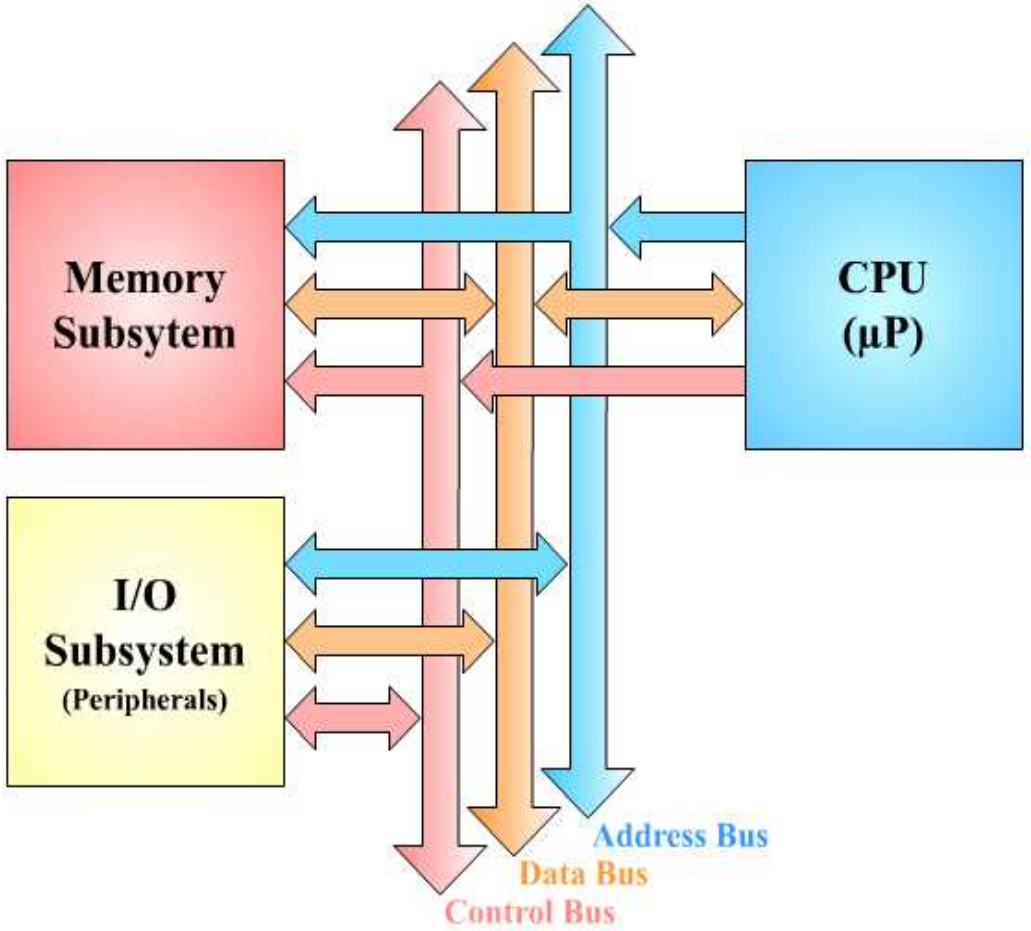
Basics of Hardware Design

We will be talking about!

- **Basic building blocks of a computer**
- **Sub-systems of CPU**
- **Processor design steps**
- **Processor design parameters**

Basic building blocks of a computer

- Central Processing Unit
- **Subsystems:**
  - Memory
  - Input / Output (Peripherals)
- Buses

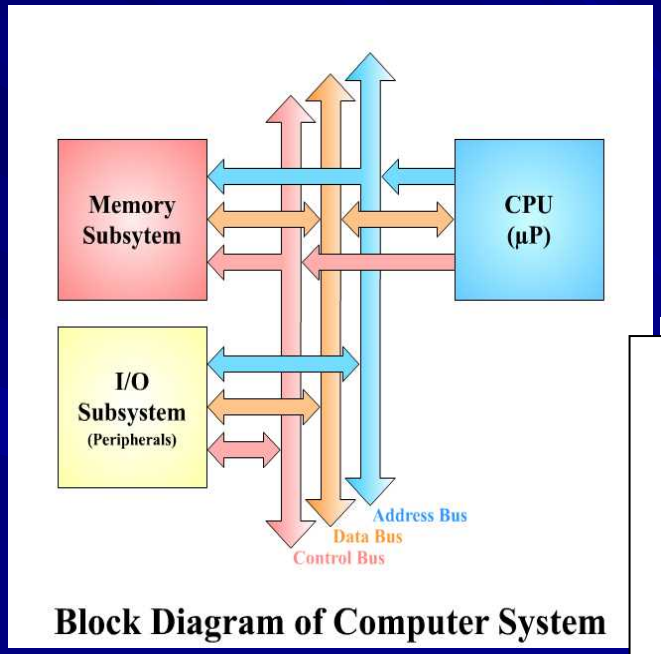


Sub-systems of Central Processing Unit

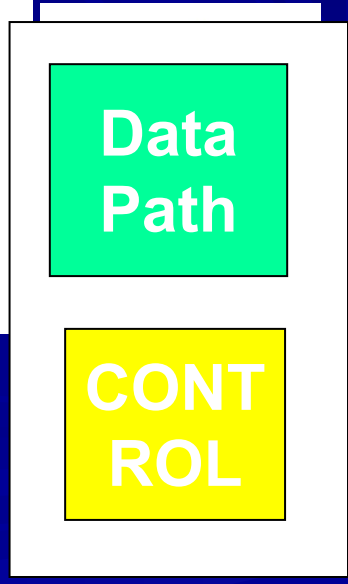
- At a “higher level” a CPU can be viewed as consisting of two sub-systems
  - **Datapath:**

    the path that facilitates the transfer of information from one part (register/memory/ IO) to the other part of the system
  - **Control:**

    the hardware that generates signals to control the sequence of steps and direct the flow of information through the datapath



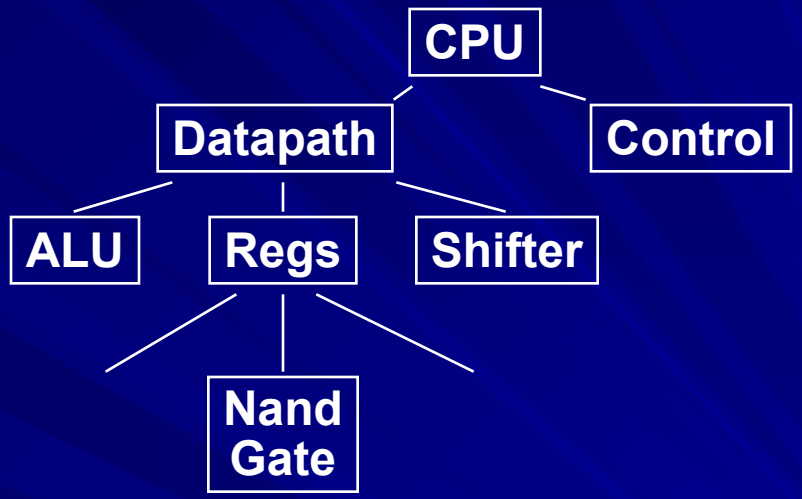
Block Diagram of Computer System



Design Process

Design is a "creative process," not a simple method

Design Finishes As Assembly



-- Design understood in terms of components and how they have been assembled

-- Top Down of complex functions (behaviors) into more primitive functions

-- bottom-up *composition* of primitive building blocks into more complex assemblies

Processor Design Steps

- **Design the Instruction Set Architecture**
- **Use RTL to describe the behavior of the processor**
  - static as well as dynamic
  - includes the functional description of each instruction in the ISA
- **Select a suitable implementation (internal organization) of the data path**
- **Map the behavioral RTL description of each instruction on to a set of structural RTL, based on the chosen implementation**
  - implies the existence of suitable timing intervals provided by synchronous clocking signals

Processor Design Steps .. Cont'd

- Prepare a list of "control signals" to be activated corresponding to each structural RTL statement
- Develop logic circuits to generate the necessary control signals
- Tie every thing together – datapath and control signals
- Other things which should be minimized
  - Amount of control hardware
  - Development time

Performing an Operation

- Each instruction of a program is performed in two phases:
  - Instruction Fetch
  - Instruction Execute
- Each phase is divided into number of steps, called Micro-operation
- A micro-operation is completed in a fixed time interval
- The number of micro-operations is determined by the datapath implementation

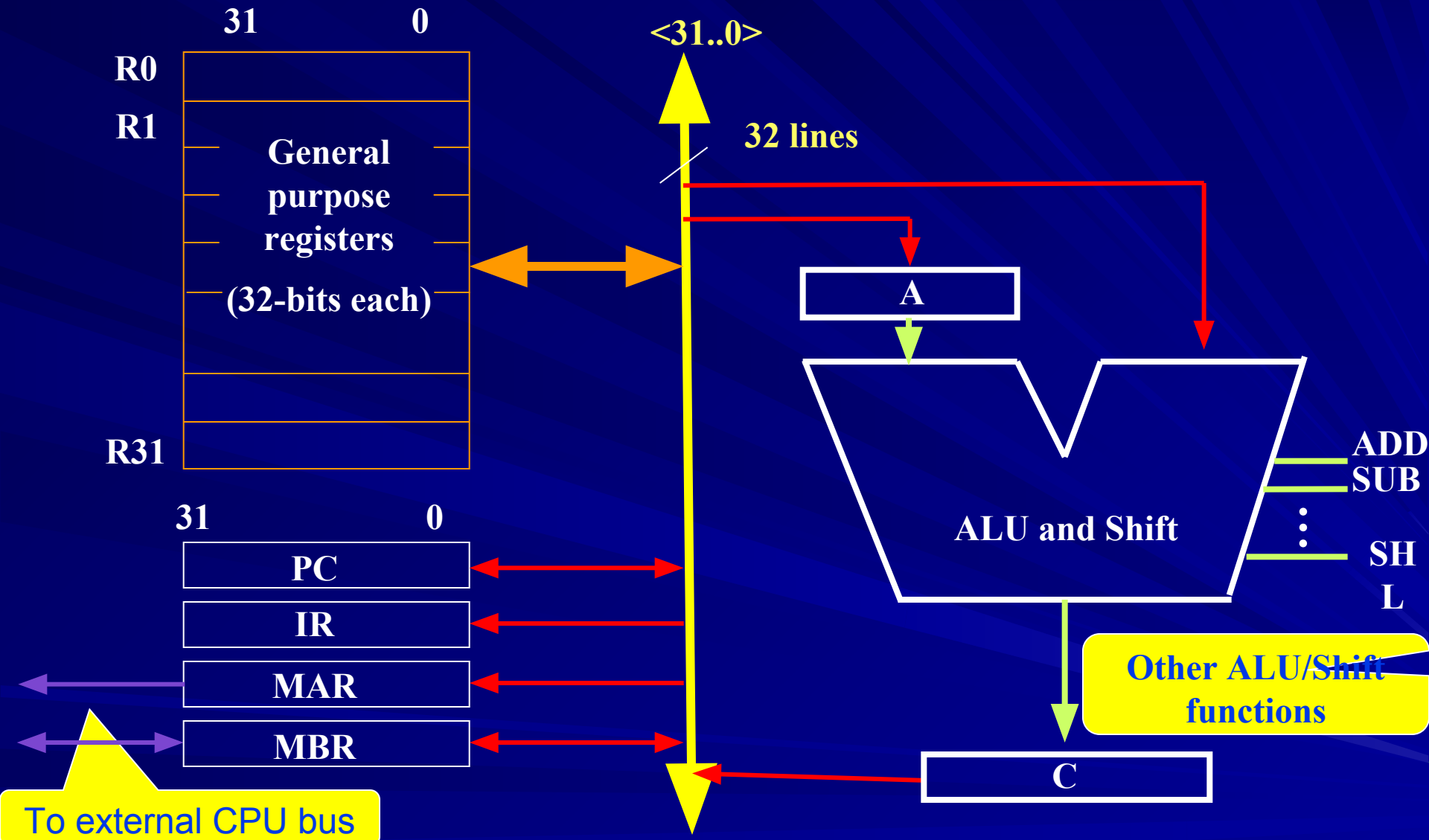
Datapath Implementations

- The ***datapath*** is the arithmetic organ of the Von- Neumann's stored-program organization
- Typically, the datapath may be implemented as:
  - Unibus structure
  - 2-bus structure
  - 3-bus structure

Datapath Implementation

- It consists of registers, internal buses, arithmetic units and shifters
- Each register in the register file has:
  - a load control line that enables data load to register
  - a set of tri-state buffers between its output and the bus
  - a read control line that enables its buffer and place the register on the bus

A Typical Unibus Datapath Implementation



Typical Unibus Datapath Structure

- It consists of a register file having 32 registers each of 32-bit and internal bus connecting the arithmetic and shifter unit to the register file
- Other registers (PC, IR, MAR, MBR, A, C) have a load control line too
- Registers PC and MBR also have a set of tri-state buffers between their output and the internal CPU bus
- Additionally, registers MAR and MBR have other circuitry connecting them to the external CPU bus

RTL micro-operations of Unibus structure

- **Instruction Fetch:**

  Completed in the following three steps (time intervals):

  T0 MAR □ PC, C □ PC + 4;

  T1 MBR □ M[MAR], PC □ C;

  T2 IR □ MBR

- **Instruction Execute:**

  Instructions of different classes are Completed in the different number of steps (time intervals):

Execution Phase micro-operations of Unibus

- **R-type Arithmetic/Logical Instructions** (Add/Sub/And/OR ra, rb, rc) or immediate

  T3 A □ R[rb];

  T4 C □ A op R[rc]; or T4 C □ A op Const(sign extended)

  T5 R[ra] □ C;

- **R-type 2-address instructions** (e.g. NOT ra, rb)

  T3 C □ NOT(R[rb]);

  T4 R[ra] □ C;

RTL micro-operations of Unibus structure

- **Load/store Instructions** **(ld/st ra, c2(rb)**

T3 A □ ((rb = 0) : 0, (rb ≠ 0): R[rb]);

T4 C □A + (sign extended and shifted c2);

T5 MAR □ C;

T6 MBR □ M[MAR]; (load) MBR □ R [ra]; (store)

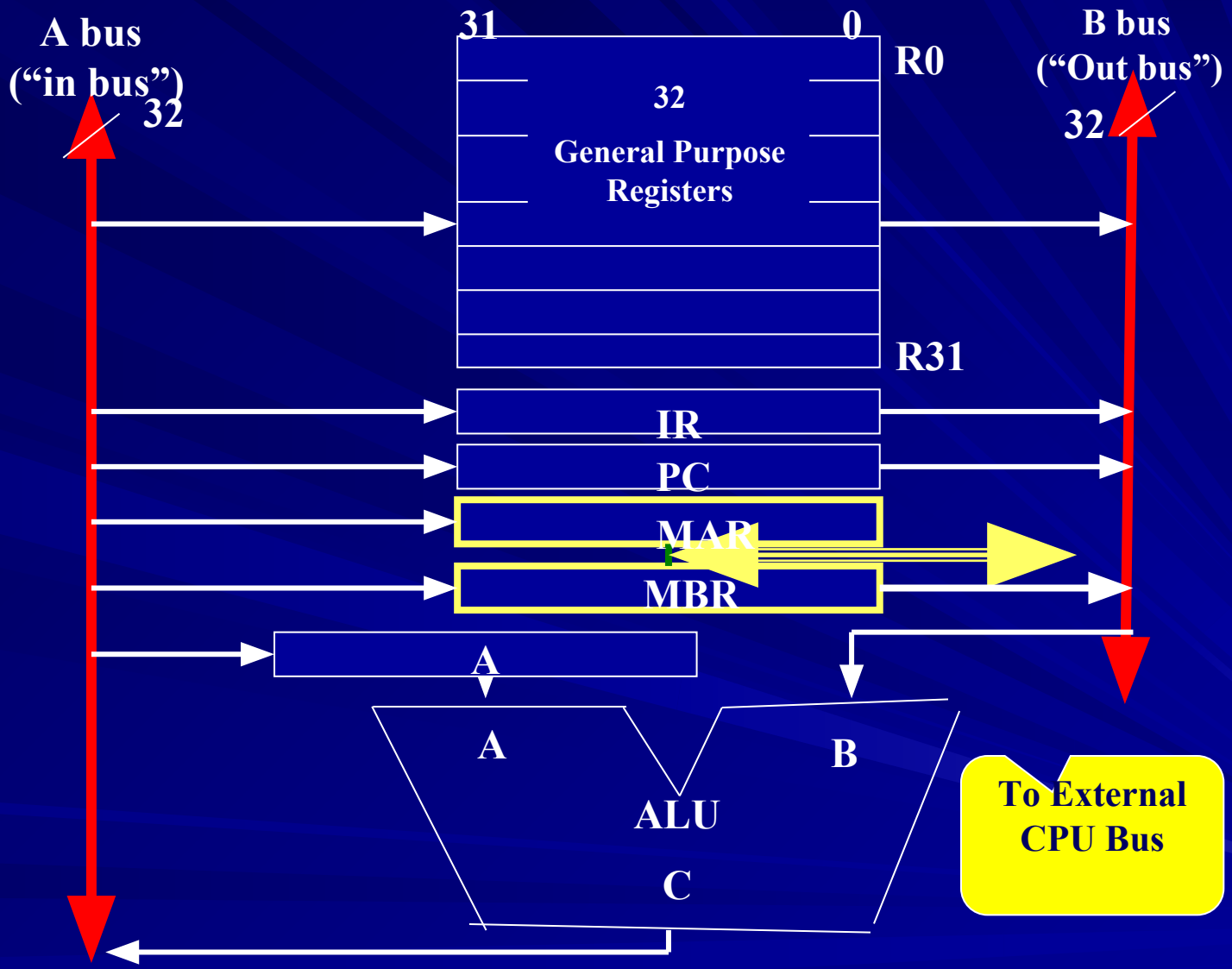
T7 R[ra] □ MBR; (load) M[MAR] □ MBR; (store)

- **Branch instructions** **(e.g. : brzr rb, rc)**

T3 CON □ cond(R[rc]);

T4 CON: PC □ R[rb];

A 2-bus implementation



Typical 2-bus Datapath Structure

- Registers and arithmetic and logic unit are identical to uni-bus structure
- The structure contains two internal buses called the in-bus and out-bus
- The in-bus carries data to be written into registers and out-bus carries data read out from the registers
- The output of ALU is directly connected to the in-bus instead of through register C as in Uni-bus structure

Fetch/Execution Phase micro-operations of 2-bus

- **Three micro-operations (steps) of the Fetch Phase are identical to Uni-bus structure except** C□ PC+4 **in step** T0
- **R-type Arithmetic/Logical Instructions are completed in two steps instead of three** (Add/Sub/And/OR ra, rb, rc) or immediate

  T3 A □ R[rb];

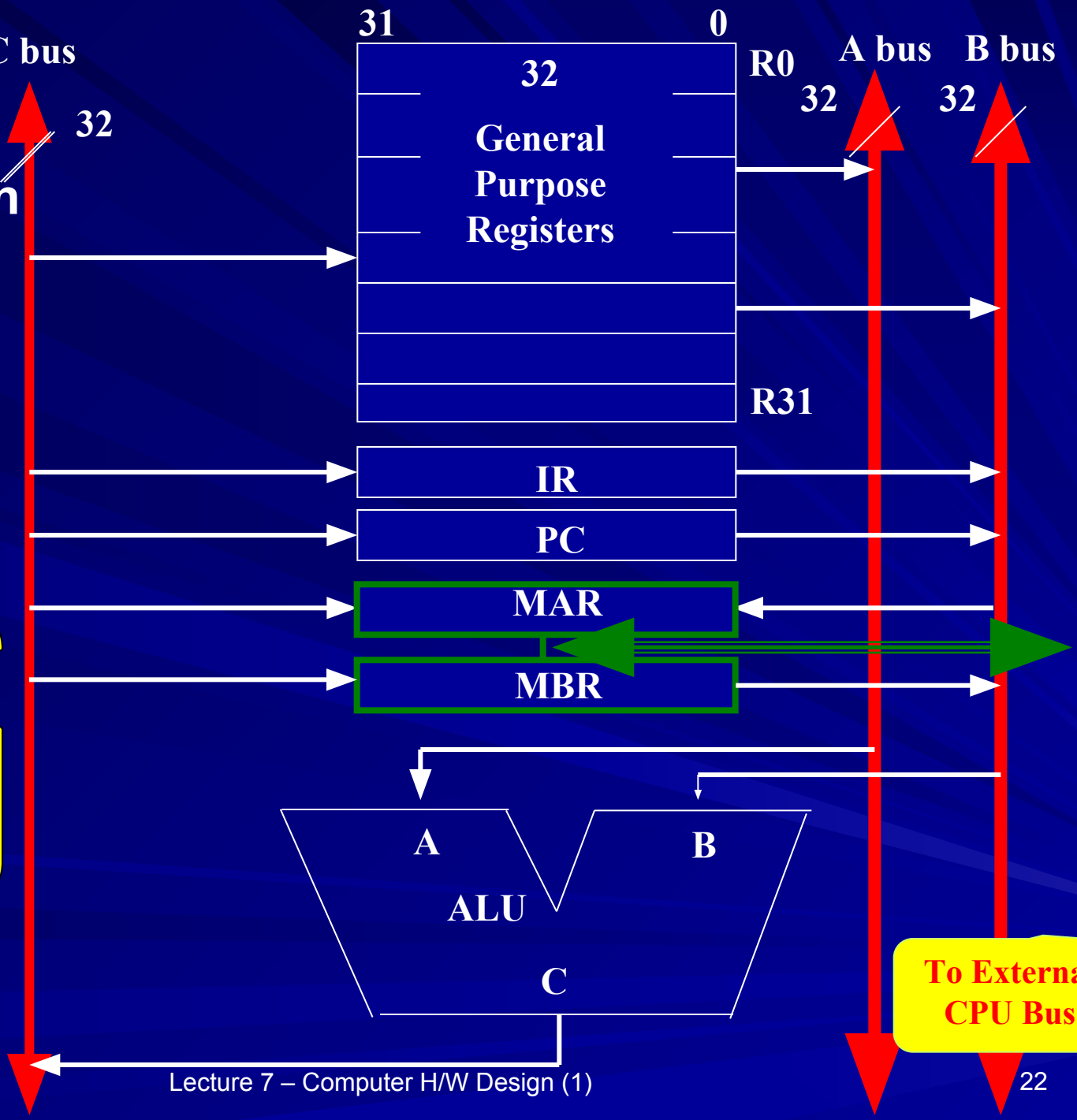
  T4 R[ra] □ A op R[rc];
- **R-type 2-address instructions** (e.g. NOT ra, rb)

  T3 R[ra] □ NOT(R[rb]);

A 3-bus implementation

All three buses are "Internal processor buses"

The register file must have 2 read ports and one write port



Typical 3-bus Datapath Structure

- Registers and arithmetic and logic unit are identical to uni-bus and 2-bus structure
- The structure contains three internal buses called the A-bus, B-bus and C-bus
- The register file contains two read ports connected to A-bus and B- bus and one write port connected to C-bus
- The registers A and C are not provided as the A input and C output of ALU are connected the bus A and C respectively
- Fetch Phase is completed in two steps and Execute phase of R-type instructions in one step

Fetch and Execute of sub instruction

using the 3-bus data path implementation

Format: sub ra, rb, rc

	Step	RTL
Instruction Fetch	T0	MAR←PC; MBR ← M[MAR], PC ← PC + 4;
	T1	IR ← MBR;
Instruction Execute	T2	R[ra] ← R[rb] - R[rc];

At the end of each sequence, the timing step generator is initialized to T0

cannot use edge-triggered FFs to implement MAR as done before

Processor Design Parameters

- Recall:

  Execution time (ET) = IC x CPI X T

- **Instruction Count = IC**
- **Clock Cycles per Instruction = CPI**
- **Clock cycle or time period = T**

- **Note that Implementation affects CPI and T**

Single and Multi cycle Datapaths

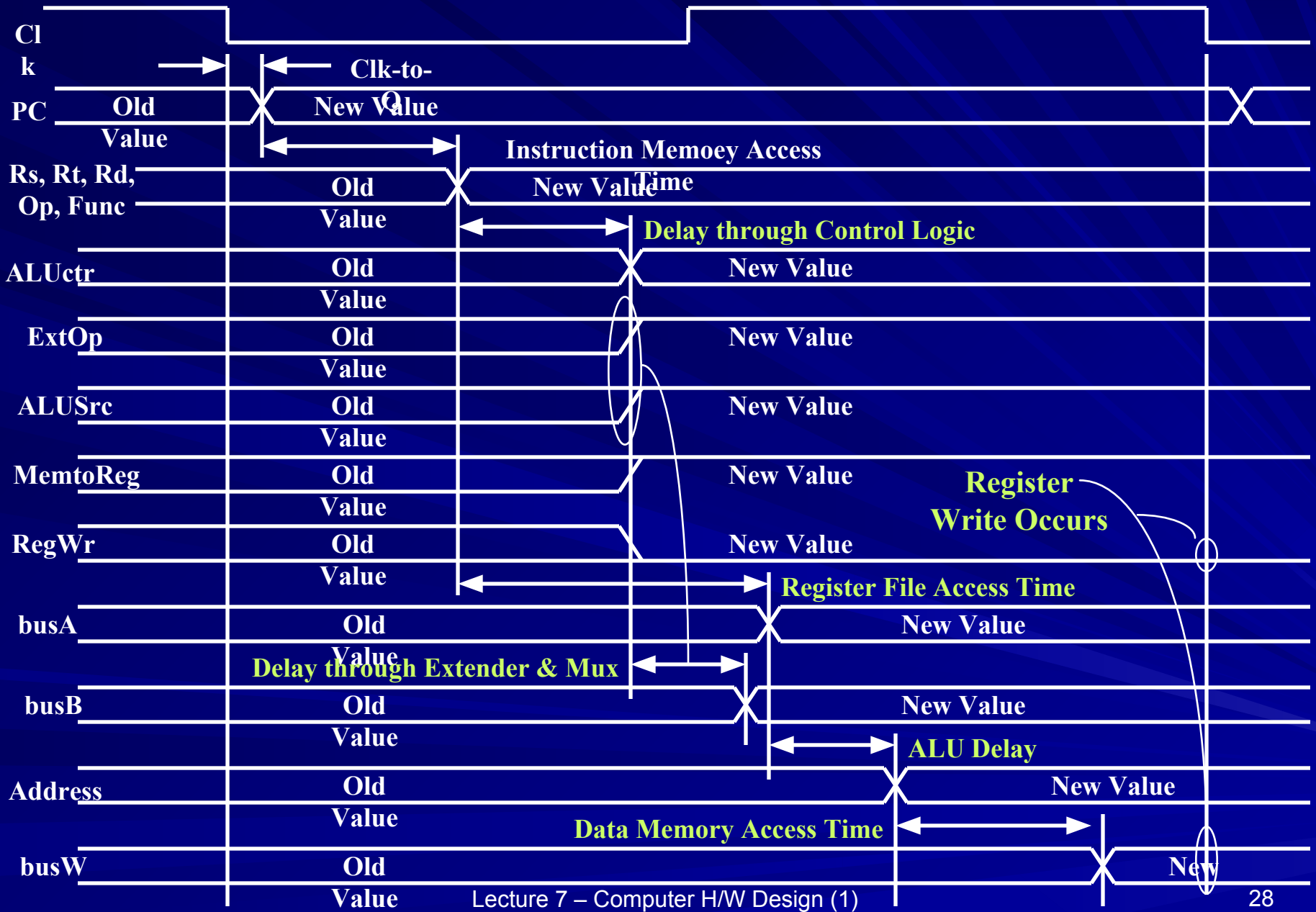
- The datapath where an instruction is fetched and executed in one clock cycle, e.g., CPI =1, is referred to as **SINGLE CYCLE** datapath

- The datapath where different classes of instruction are fetched and executed in variable number of cycles is referred to as **MULTI-CYCLE** datapath

Single cycle Datapath

- The instruction fetch and execute phases are completed in one clock cycle
- A clock cycle is divided in to number of steps to complete the operations
- The cycle length is constant whereas number of steps (or micro operation) may be variable
- The timing step generator returns to T0 on the completion of a cycle

Worst Case Timing (Load)



Single cycle Timing

- This timing diagram shows the worst case timing of single cycle datapath (which occurs at the load instruction).
- Clock-to-Q time after the clock tick, PC will present its new value to the Instruction memory.
- After a delay of instruction access time, the instruction bus (Rs, Rt, ...) becomes valid.

Single cycle Timing

- Then three things happens in parallel:

  (a) First the Control generates the control signals (Delay through Control Logic).

  (b) Secondly, the regiser file is access to put Rs onto busA.

  (c) Thirdly, in case of memory reference or immediate data instructions, we have to sign extended the immediate field to get the second operand (busB)

Single cycle Timing

- Here we assume register file access takes longer time than doing the sign extension so we have to wait until busA valid before the ALU can start the address calculation (ALU delay).
- With the address ready, we access the data memory and after a delay of the Data Memory Access time, busW will be valid.
- And by this time, the control unit would have set the RegWr signal to one so at the next clock tick, we will write the new data coming from memory (busW) into the register file.

Single cycle Memory Structure

- As clear from the timing diagram, the memory address (from PC) for instruction fetch; and from ALU for the data read/write; are available on the bus simultaneously – thus gives rise to **structural hazard**

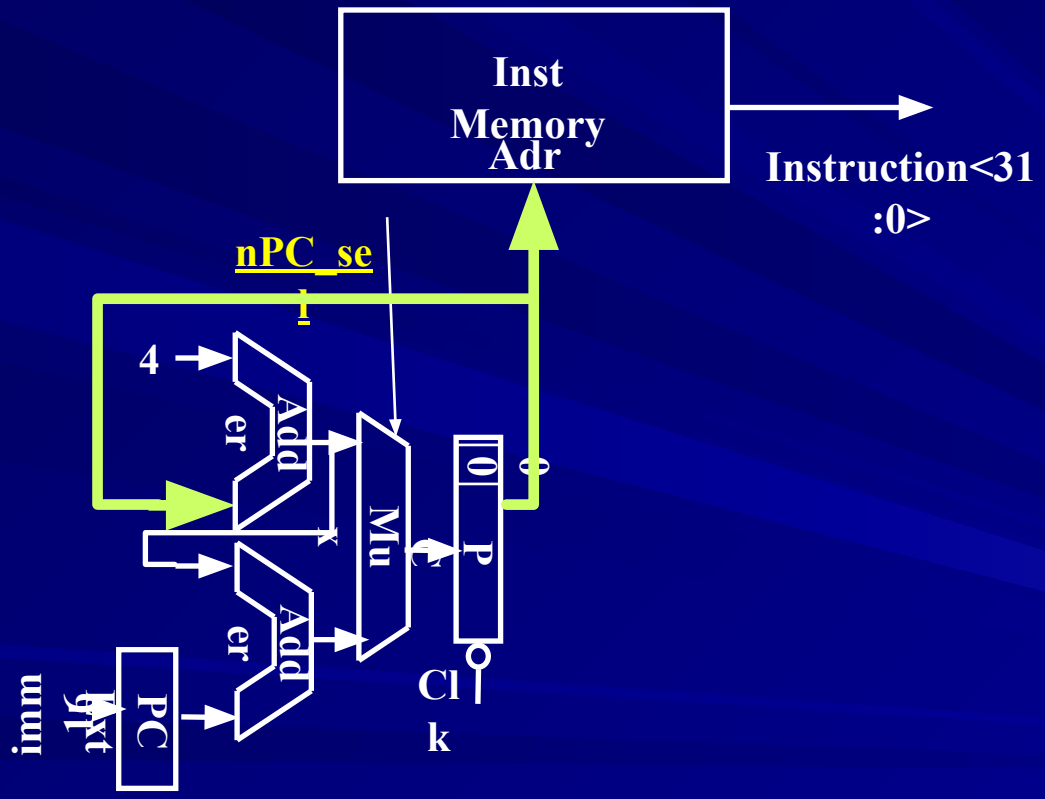
- To overcome this problem memory unit is partitioned in to parts
  - **Instruction memory**
  - **Data memory**

Single Cycle Instruction Fetch Unit

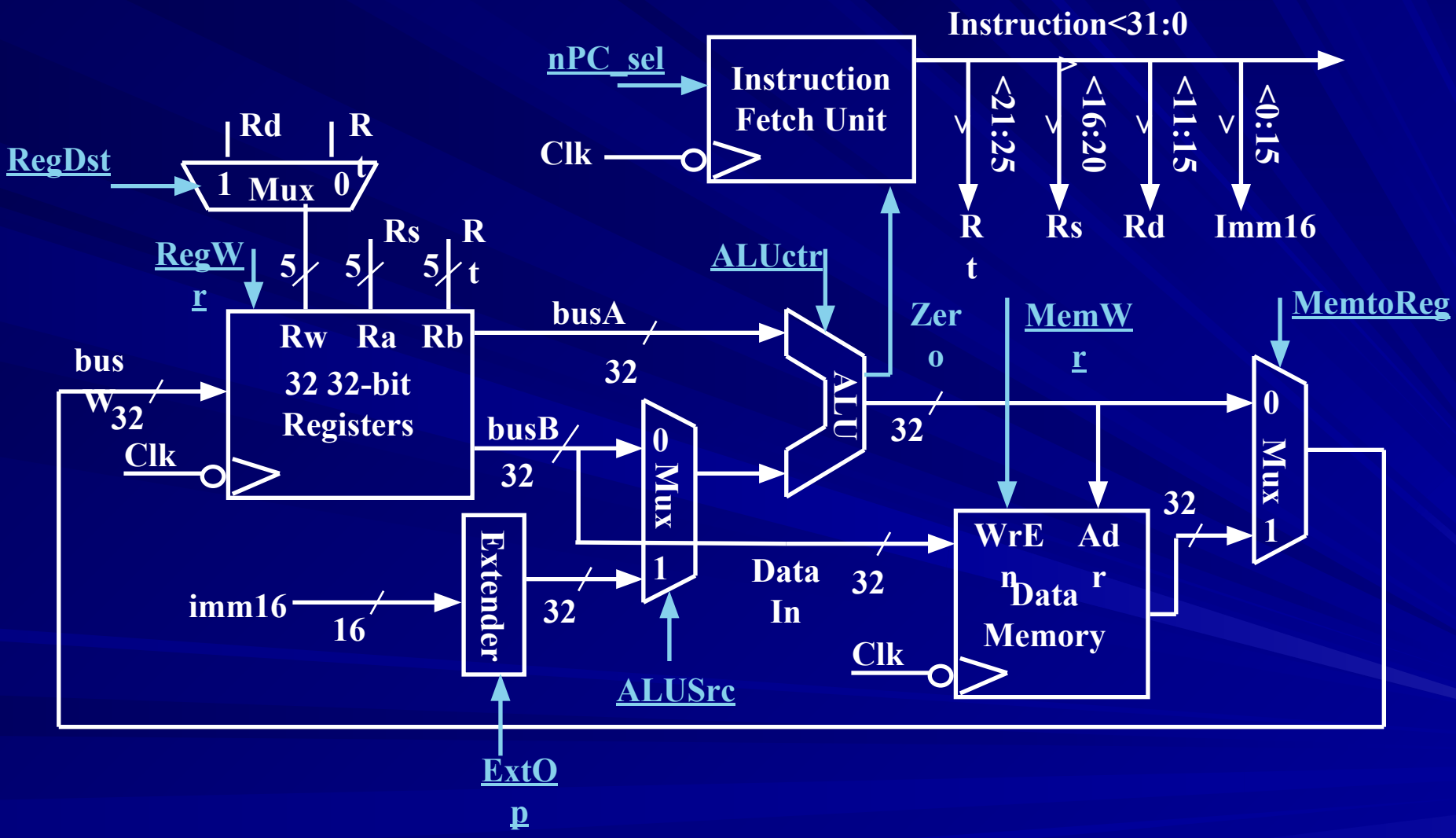
- **Fetch the instruction from Instruction memory:**

  Instruction Mem[PC]

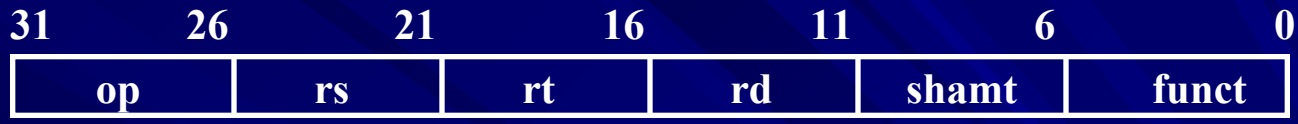
  - **This is the same for all instructions**



A Single Cycle Datapath



The Single Cycle Datapath during `Add`



- R[rd] <- R[rs] + R[rt]

The Single Cycle Datapath during Add

- This picture shows the activities at the main datapath during the execution of the Add or Subtract instructions.
- The active parts of the datapath are shown in different color as well as thicker lines.
- First of all, the Rs and Rt of the instructions are fed to the Ra and Rb address ports of the register file and cause the contents of registers specified by the Rs and Rt fields to be placed on busA and busB, respectively.
- With the ALUctr signals set to either Add or Subtract, the ALU will perform the proper operation and with MemtoReg set to 0, the ALU output will be placed onto busW.

The Single Cycle Datapath during Add

- The control we are going to design will also set RegWr to 1 so that the result will be written to the register file at the end of the cycle.

- Notice that ExtOp is don't care because the Extender in this case can either do a SignExt or ZeroExt. We DON'T care because ALUSrc will be equal to 0--we are using busB.

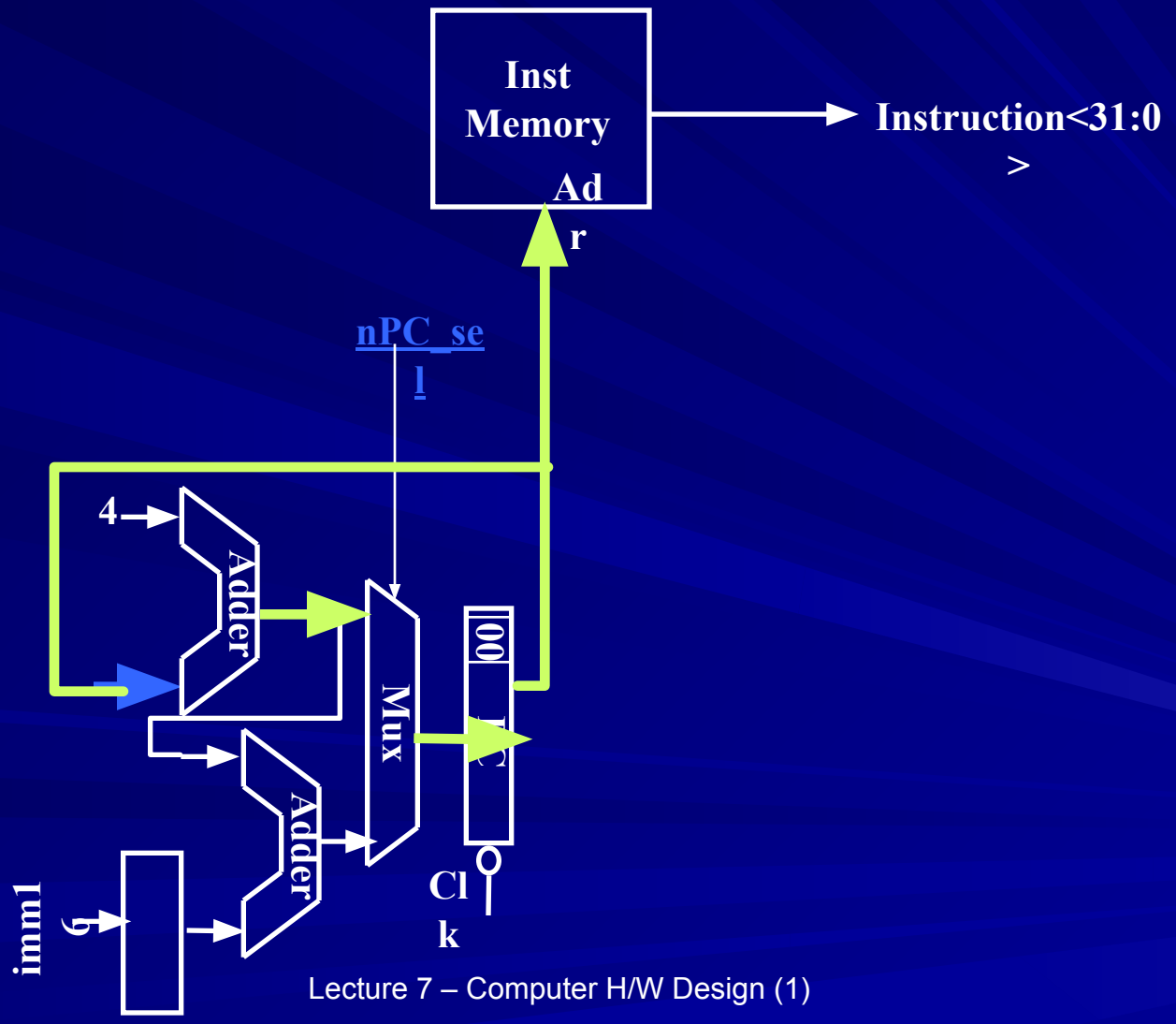
- The other control signals we need to worry about are:

  (a) MemWr has to be set to zero because we do not want to write the memory.

  (b) And Branch and Jump, we have to set to zero. Let me show you why.

Instruction Fetch Unit at the End of Add

- PC <- PC + 4; This is the same for all instructions except: Branch and Jump



Instruction Fetch Unit at the End of Add

- This picture shows the control signals setting for the Instruction Fetch Unit at the end of the Add or Subtract instruction.
- Both the Branch and Jump signals are set to 0.
- Consequently, the output of the first adder, which implements PC plus 1, is selected through the two 2-to-1 mux and got placed into the input of the Program Counter register.

Instruction Fetch Unit at the End of Add

- The Program Counter is updated to this new value at the next clock tick.
- Notice that the Program Counter is updated at every cycle. Therefore it does not have a Write Enable signal to control the write.
- Also, this picture is the same for or all instructions other than Branch and Jump.
- Therefore I will only show this picture again for the Branch and Jump instructions and will not repeat this for all other instructions.