

CS3350B
Computer Architecture
Winter 2015

**Lecture 5.6: Single-Cycle CPU:
Datapath Control (Part 1)**

Marc Moreno Maza

www.csd.uwo.ca/Courses/CS3350b

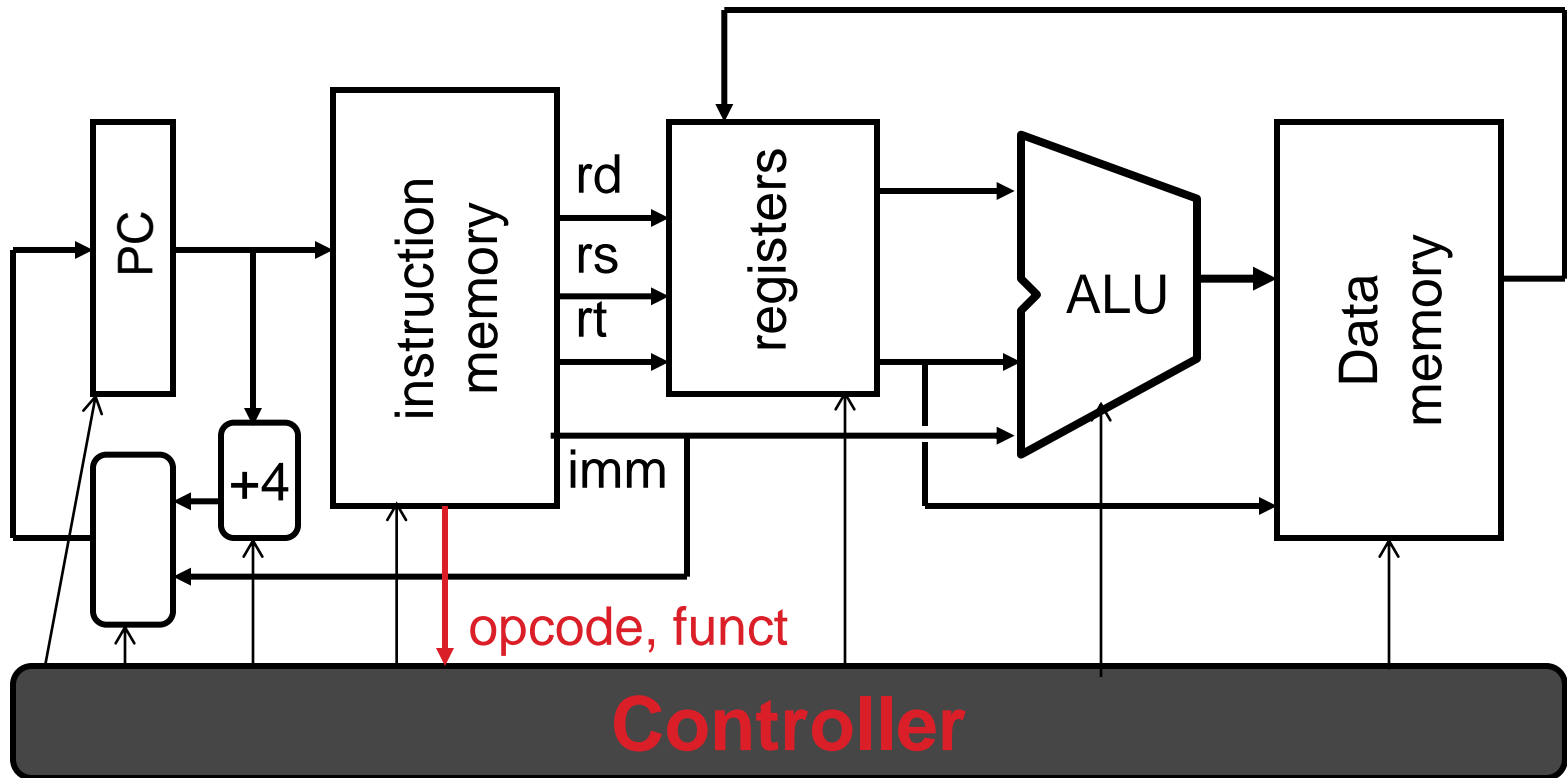
[Adapted from lectures on
Computer Organization and Design,
Patterson & Hennessy, 5th edition, 2013]

Review

- CPU design involves Datapath, Control
 - 5 Stages for MIPS Instructions
 1. Instruction Fetch
 2. Instruction Decode & Register Read
 3. ALU (Execute)
 4. Memory
 5. Register Write
- Datapath timing: single long clock cycle or one short clock cycle per stage

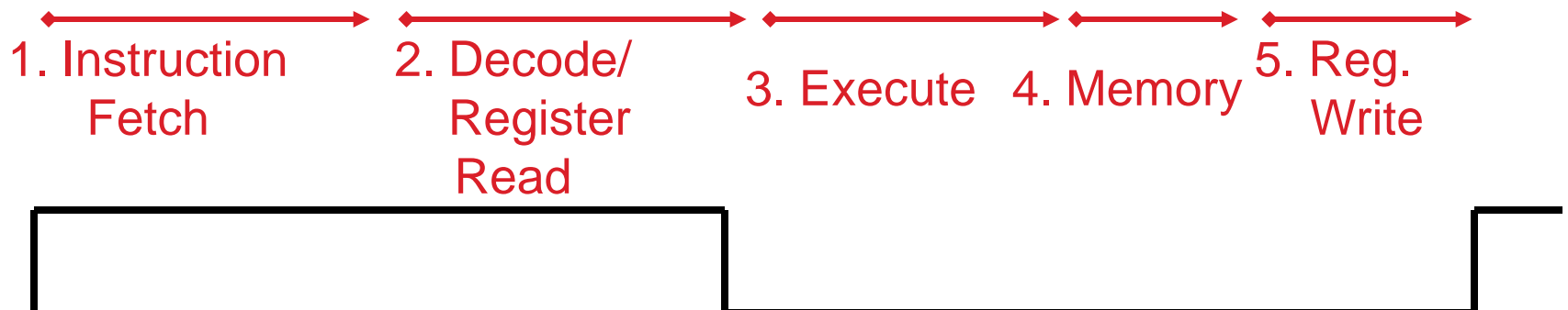
Datapath and Control

- Datapath based on data transfers required to perform instructions
- Controller causes the right transfers to happen



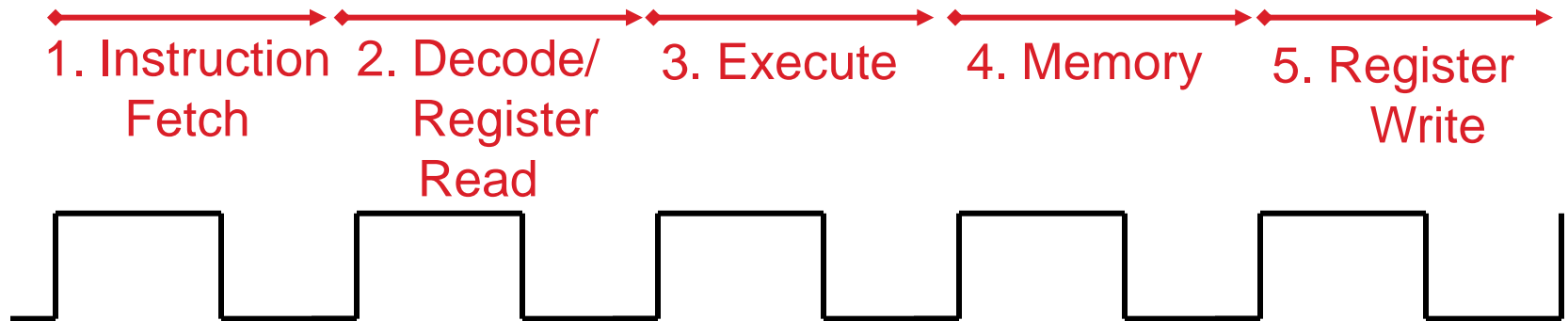
CPU Clocking (1/2)

- For each instruction, how do we control the flow of information through the datapath?
- Single Cycle CPU: All stages of an instruction completed within one long clock cycle
 - Clock cycle sufficiently long to allow each instruction to complete all stages without interruption within one cycle



CPU Clocking (2/2)

- Alternative multiple-cycle CPU: only one stage of instruction per clock cycle
 - Clock is made as long as the slowest stage

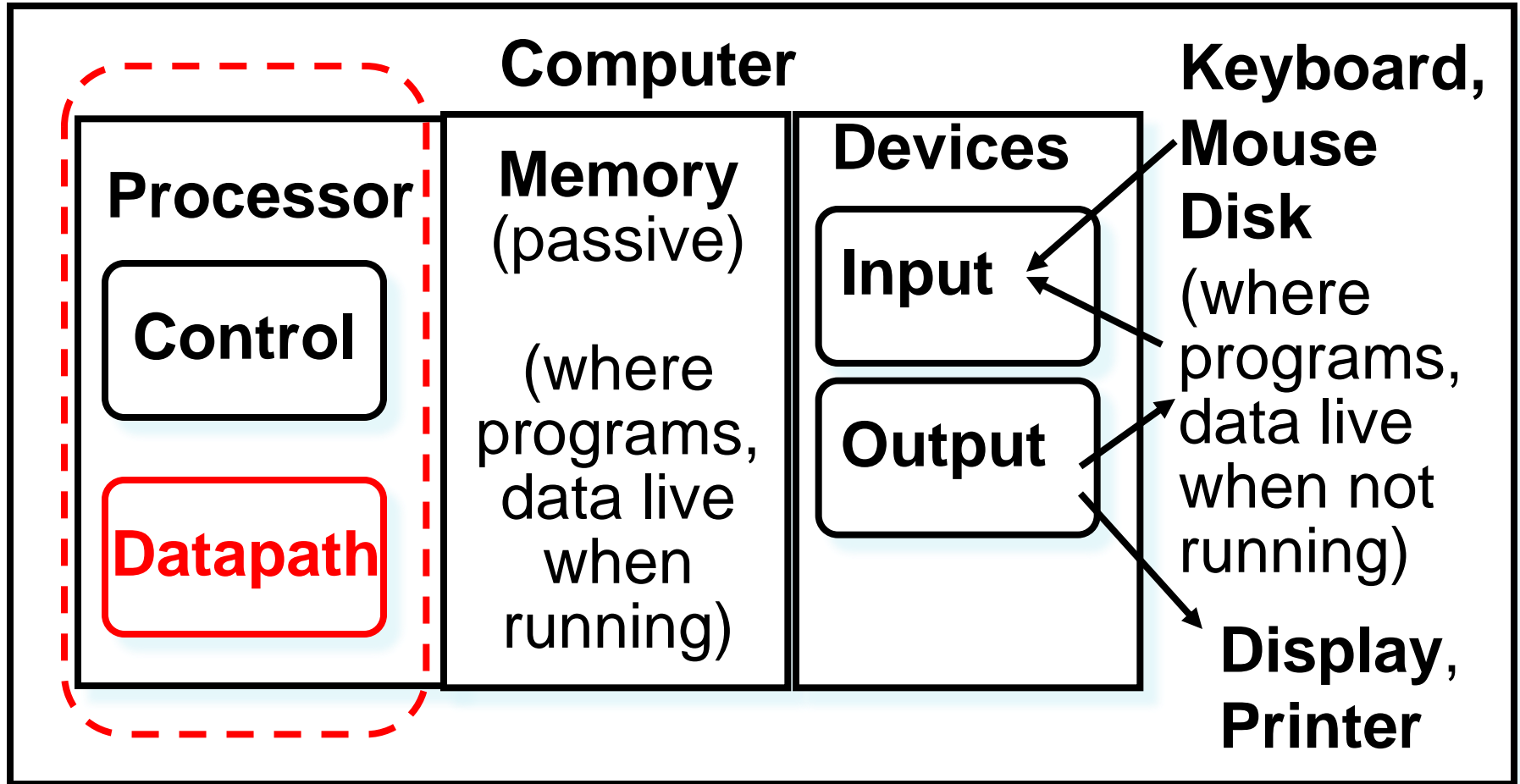


- Several significant advantages over single cycle execution:
 - Unused stages in a particular instruction can be skipped
 - OR instructions can be pipelined (overlapped)

Plan

- Stages of the Datapath
- Datapath Instruction Walkthroughs
- Datapath Design

Five Components of a Computer



Processor Design: 5 steps

Step 1: Analyze instruction set to determine datapath requirements

- Meaning of each instruction is given by register transfers
- Datapath must include storage element for ISA registers
- Datapath must support each register transfer

Step 2: Select set of datapath components & establish clock methodology

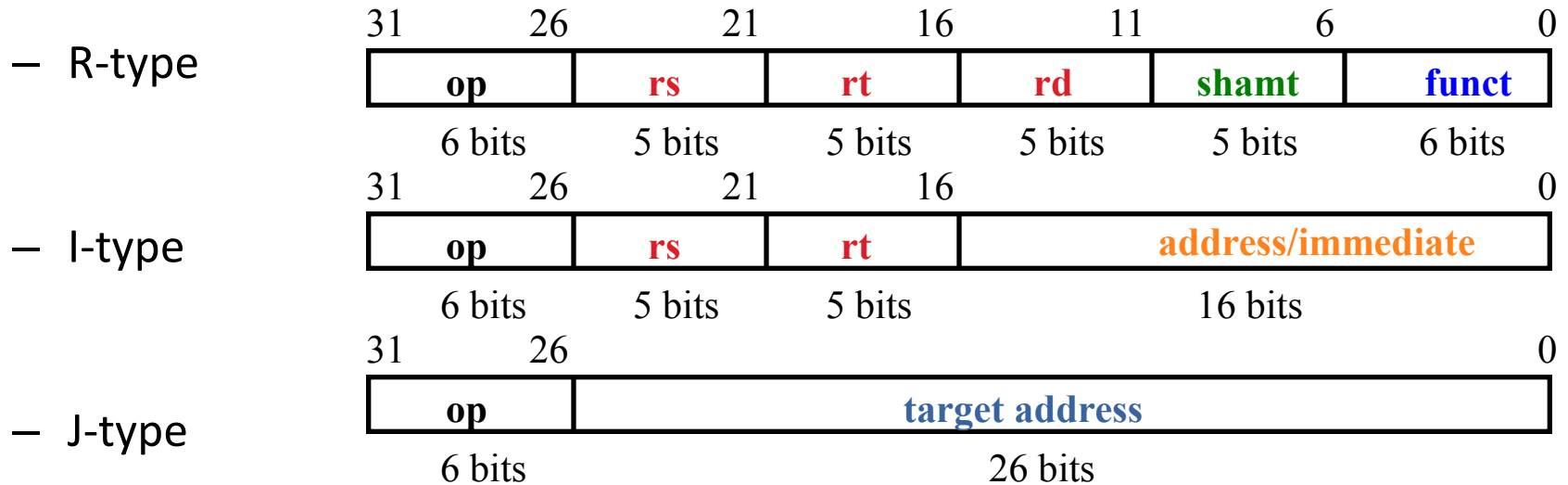
Step 3: Assemble datapath components that meet the requirements

Step 4: Analyze implementation of each instruction to determine setting of control points that realizes the register transfer

Step 5: Assemble the control logic

The MIPS Instruction Formats

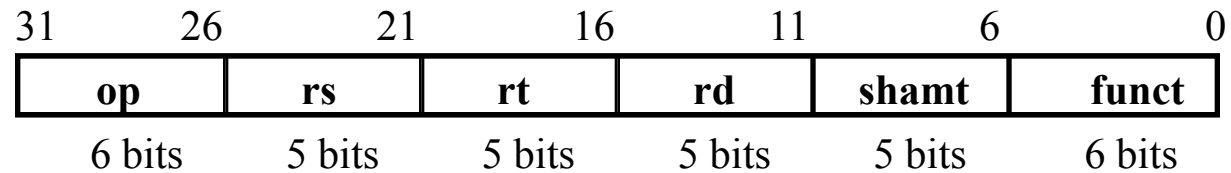
- All MIPS instructions are 32 bits long. 3 formats:



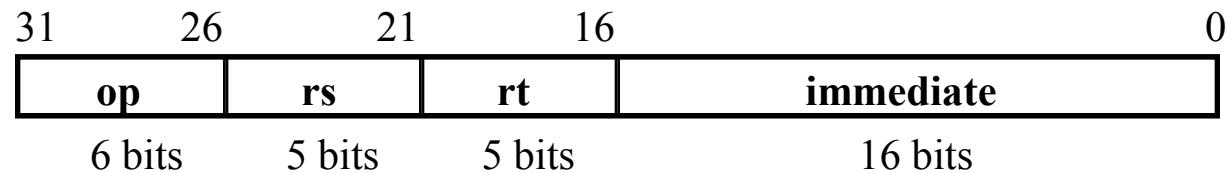
- The different fields are:
 - **op**: operation (“opcode”) of the instruction
 - **rs, rt, rd**: the source and destination register specifiers
 - **shamt**: shift amount
 - **funct**: selects the variant of the operation in the “op” field
 - **address / immediate**: address offset or immediate value
 - **target address**: target address of jump instruction

The MIPS-lite Subset

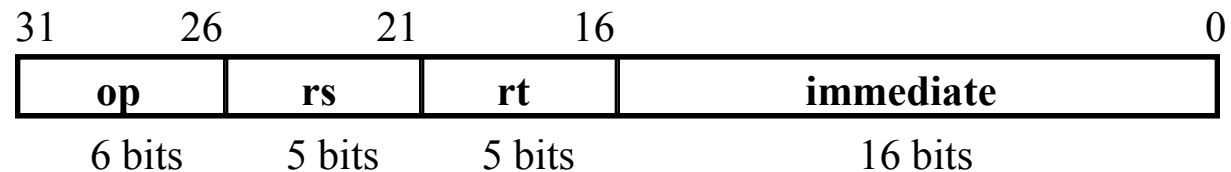
- ADDU and SUBU
 - `addu rd,rs,rt`
 - `subu rd,rs,rt`



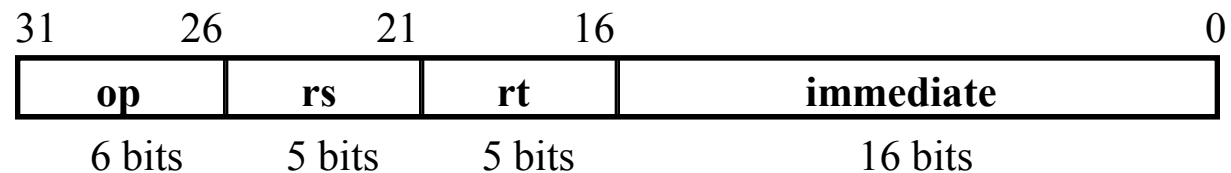
- OR Immediate:
 - `ori rt,rs,imm16`



- LOAD and STORE Word
 - `lw rt,rs,imm16`
 - `sw rt,rs,imm16`



- BRANCH:
 - `beq rs,rt,imm16`



Register Transfer Language (RTL)

RTL gives the meaning of the instructions

All start by fetching the instruction

```
{op , rs , rt , rd , shamt , funct} ← MEM[ PC ]
```

```
{op , rs , rt , Imm16} ← MEM[ PC ]
```

Inst Register Transfers

```
ADDU    R[rd] ← R[rs] + R[rt]; PC ← PC + 4
```

```
SUBU    R[rd] ← R[rs] - R[rt]; PC ← PC + 4
```

```
ORI     R[rt] ← R[rs] | zero_ext(Imm16); PC ← PC + 4
```

```
LOAD    R[rt] ← MEM[ R[rs] + sign_ext(Imm16) ]; PC ← PC + 4
```

```
STORE   MEM[ R[rs] + sign_ext(Imm16) ] ← R[rt]; PC ← PC + 4
```

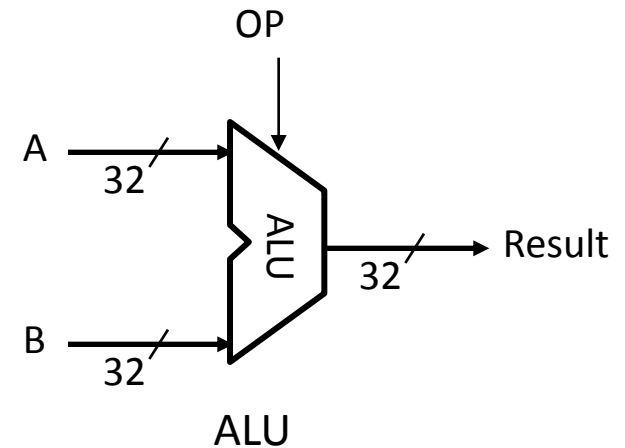
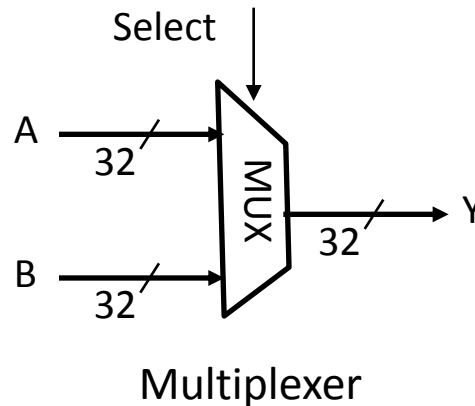
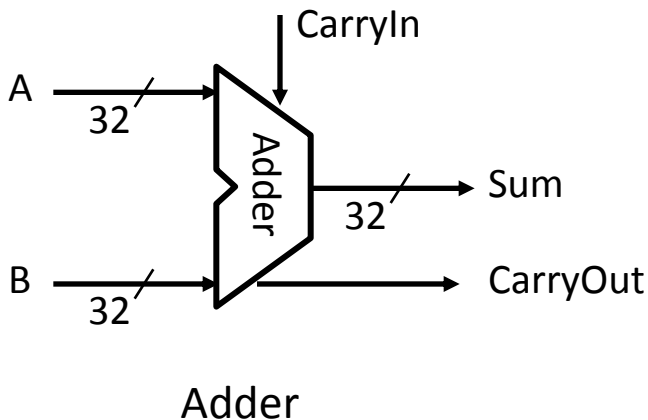
```
BEQ     if ( R[rs] == R[rt] )  
         then PC ← PC + 4 + (sign_ext(Imm16) || 00)  
         else PC ← PC + 4
```

Step 1: Requirements of the Instruction Set

- Memory (MEM)
 - Instructions & data (will use one for each)
- Registers (R: 32 x 32)
 - Read RS
 - Read RT
 - Write RT or RD
- PC
- Extender (sign/zero extend)
- Add/Sub/OR unit for operation on register(s) or extended immediate
- Add 4 (+ maybe extended immediate) to PC
- Compare registers?

Step 2: Components of the Datapath

- Combinational Elements
- Storage Elements + Clocking Methodology
- Building Blocks



ALU Needs for MIPS-lite + Rest of MIPS

- Addition, subtraction, logical OR, ==:

ADDU $R[rd] = R[rs] + R[rt]; \dots$

SUBU $R[rd] = R[rs] - R[rt]; \dots$

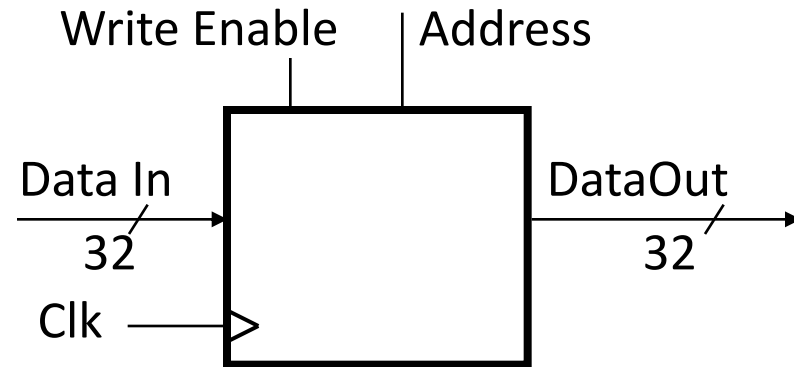
ORI $R[rt] = R[rs] | \text{zero_ext}(\text{Imm16}) \dots$

BEQ $\text{if} (R[rs] == R[rt]) \dots$

- Test to see if output == 0 for any ALU operation gives == test. How?
- P&H also adds AND, Set Less Than (1 if $A < B$, 0 otherwise)
- ALU follows Chapter 5

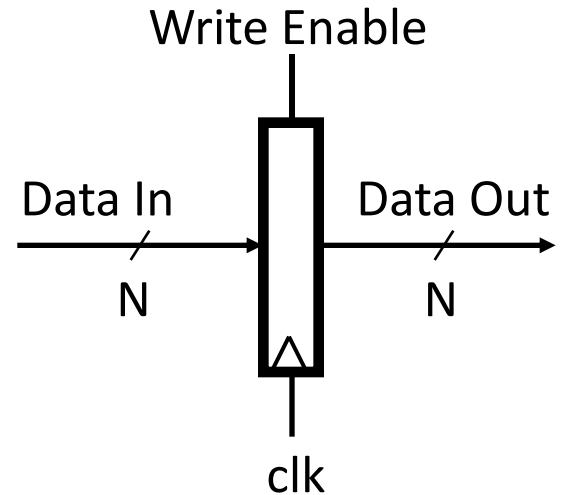
Storage Element: Idealized Memory

- Memory (idealized)
 - One input bus: Data In
 - One output bus: Data Out
- Memory word is found by:
 - Address selects the word to put on Data Out
 - Write Enable = 1: address selects the memory word to be written via the Data In bus
- Clock input (CLK)
 - CLK input is a factor ONLY during write operation
 - During read operation, behaves as a combinational logic block: Address valid \Rightarrow Data Out valid after “access time”



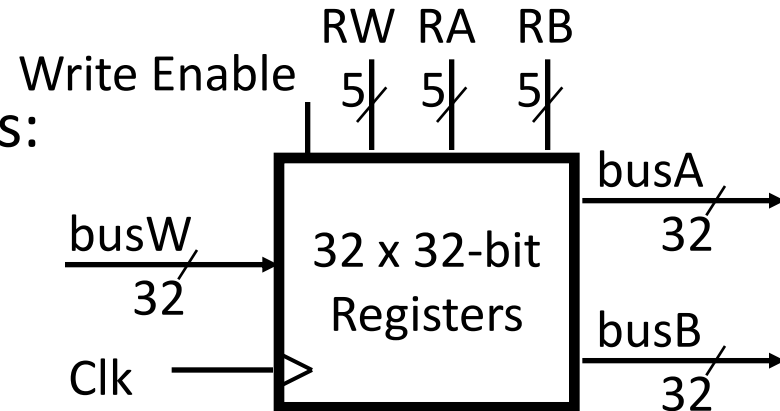
Storage Element: Register (Building Block)

- Similar to D Flip Flop except
 - N-bit input and output
 - Write Enable input
- Write Enable:
 - Negated (or deasserted) (0): Data Out will not change
 - Asserted (1): Data Out will become Data In on positive edge of clock



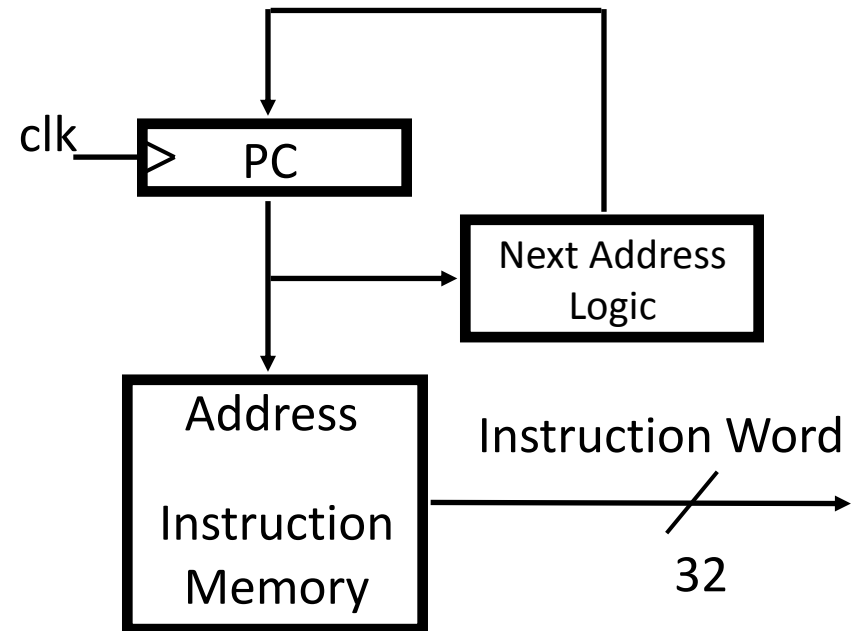
Storage Element: Register File

- Register File consists of 32 registers:
 - Two 32-bit output busses: busA and busB
 - One 32-bit input bus: busW
- Register is selected by:
 - RA (number) selects the register to put on busA (data)
 - RB (number) selects the register to put on busB (data)
 - RW (number) selects the register to be written via busW (data) when Write Enable is 1
- Clock input (clk)
 - Clk input is a factor ONLY during write operation
 - During read operation, behaves as a combinational logic block:
 - RA or RB valid \Rightarrow busA or busB valid after “access time.”



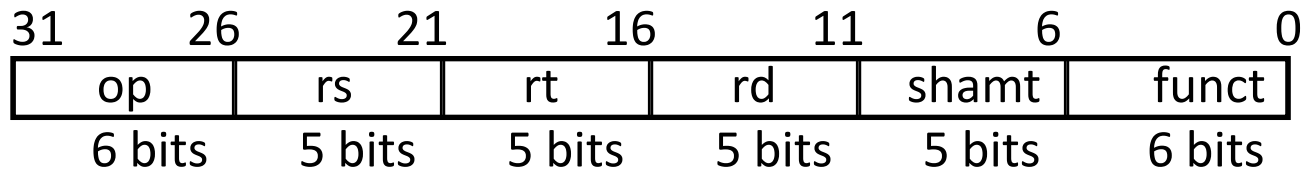
Step 3a: Instruction Fetch Unit

- Register Transfer Requirements
⇒ Datapath Assembly
- Instruction Fetch
- Read Operands and Execute Operation
- Common RTL operations
 - Fetch the Instruction:
 $\text{mem}[\text{PC}]$
 - Update the program counter:
 - Sequential Code:
 $\text{PC} \leftarrow \text{PC} + 4$
 - Branch and Jump:
 $\text{PC} \leftarrow \text{“something else”}$

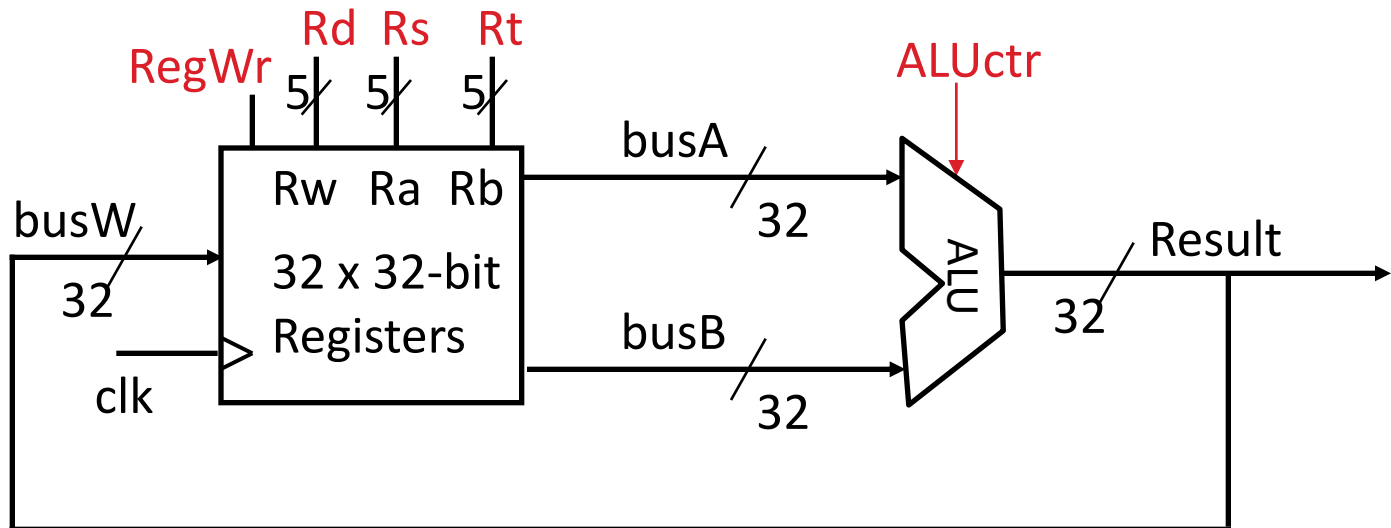


Step 3b: Add & Subtract

- $R[rd] = R[rs] \text{ op } R[rt]$ (**addu rd, rs, rt**)
 - Ra, Rb, and Rw come from instruction's Rs, Rt, and Rd fields

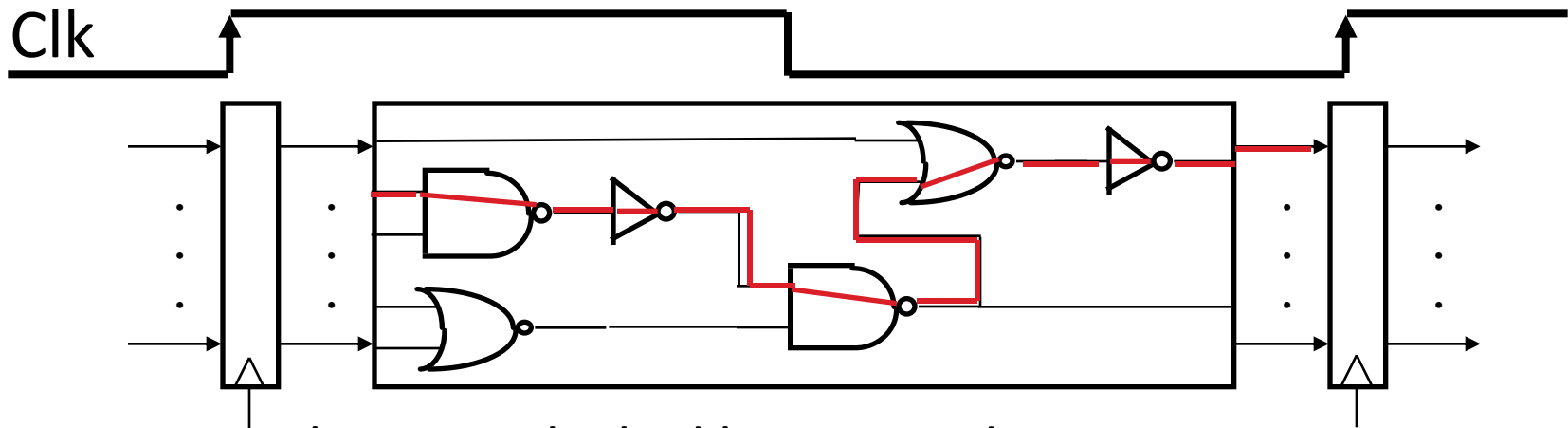


- **ALUctr** and **RegWr**: control logic after decoding the instruction



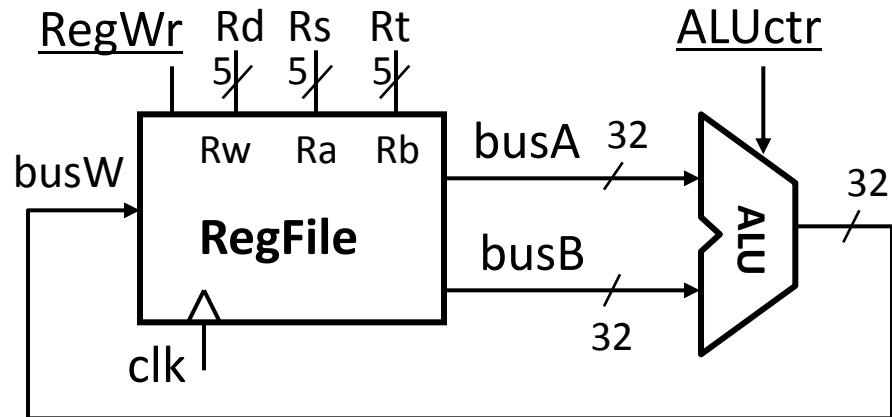
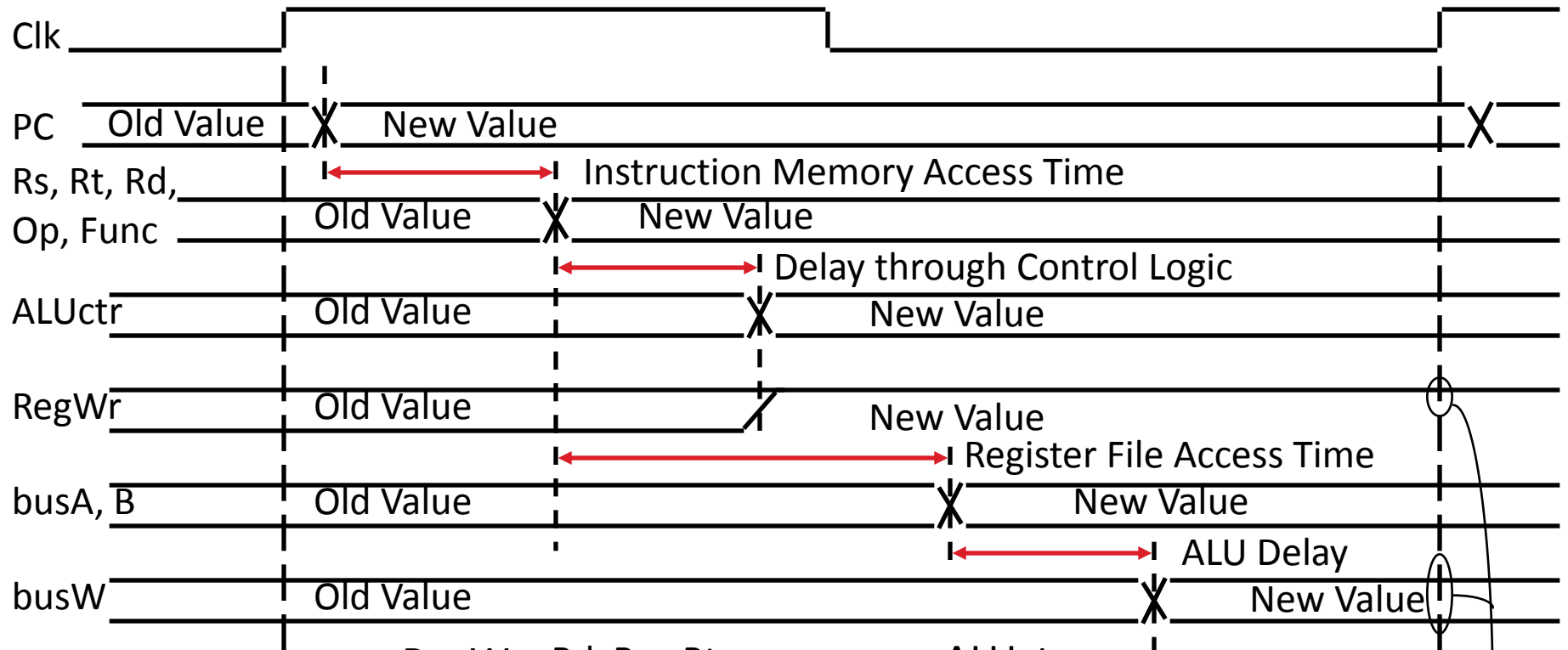
- ... Already defined the register file & ALU

Clocking Methodology



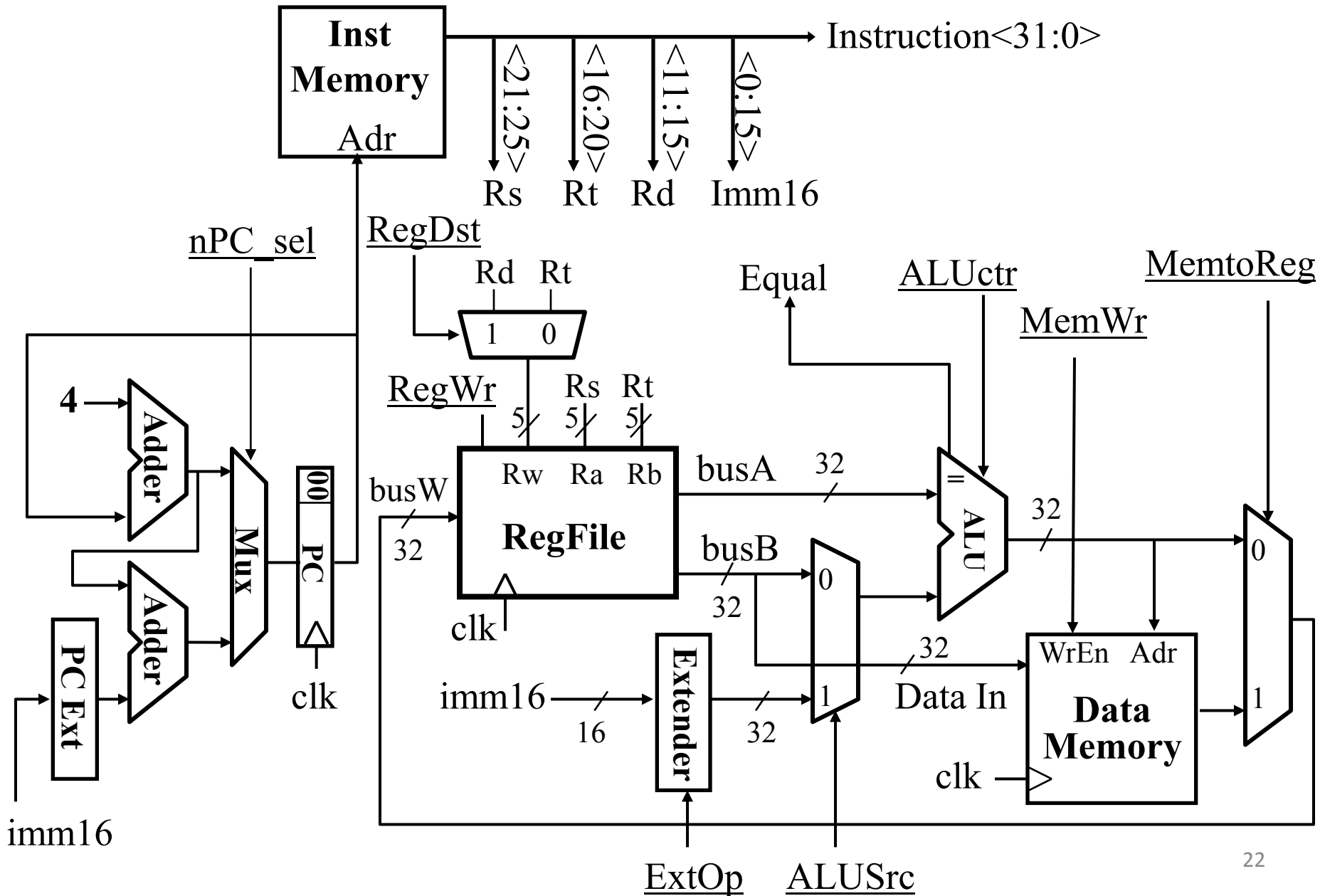
- Storage elements clocked by same edge
- Flip-flops (FFs) and combinational logic have some delays
 - Gates: delay from input change to output change
 - Signals at FF D input must be stable before active clock edge to allow signal to travel within the FF (set-up time), and we have the usual clock-to-Q delay
- “Critical path” (longest path through logic) determines length of clock period

Register-Register Timing: One Complete Cycle



Register Write Occurs Here

Putting it All Together: A Single Cycle Datapath



Processor Design: 3 of 5 steps

Step 1: Analyze instruction set to determine datapath requirements

- Meaning of each instruction is given by register transfers
- Datapath must include storage element for ISA registers
- Datapath must support each register transfer

Step 2: Select set of datapath components & establish clock methodology

Step 3: Assemble datapath components that meet the requirements

Step 4: Analyze implementation of each instruction to determine setting of control points that realizes the register transfer

Step 5: Assemble the control logic