

# Designing MIPS Processor (Single-Cycle)

Presentation G

**Reading Assignment: 5.1-5.4**

Slides by Gojko Babić

## Introduction

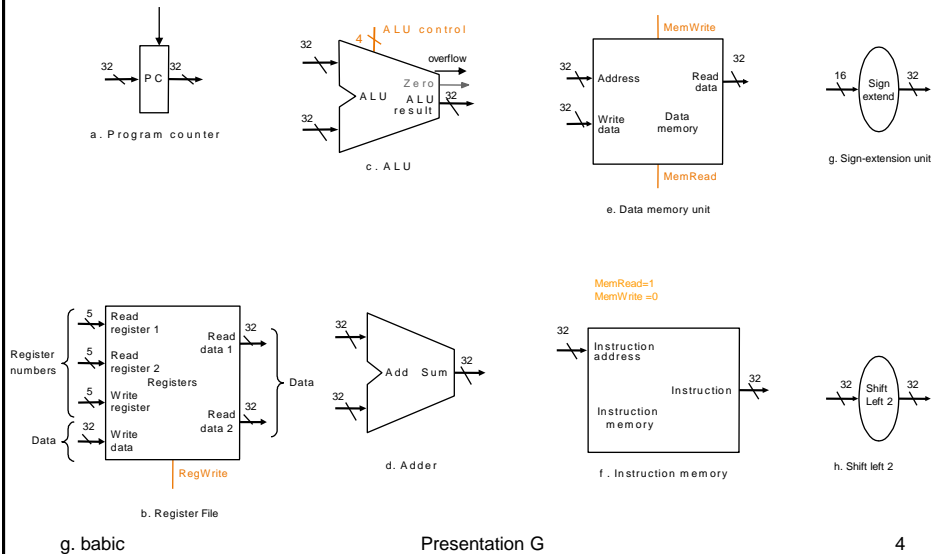
---

- We're now ready to look at an implementation of the system that includes MIPS processor and memory.
- The design will include support for execution of only:
  - memory-reference instructions: **lw & sw**,
  - arithmetic-logical instructions: **add, sub, and, or, slt & nor**,
  - control flow instructions: **beq & j**,
  - exception handling: **illegal instruction & overflow**.
- But that design will provide us with principles, so many more instructions could be easily added such as: **addu, lb, lbu, lui, addi, adiu, sltu, slti, andi, ori, xor, xori, jal, jr, jalr, bne, beqz, bgtz, bltz, nop, mfhi, mflo, mfepc, mfco, lwc1, swc1, etc.**

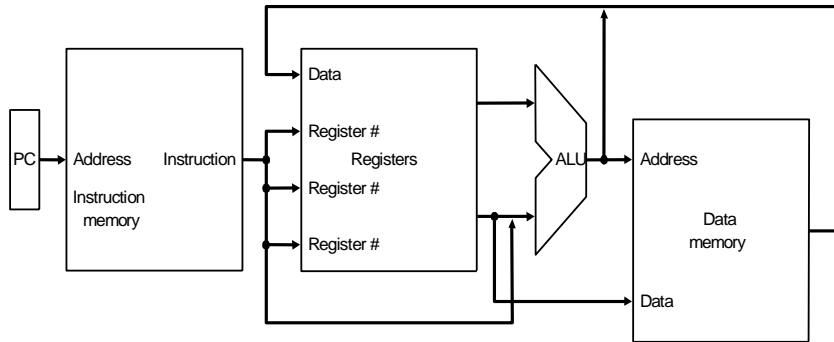
# Single Cycle Design

- We will first design a simpler processor that executes each instruction in only **one clock cycle time**.
- This is not efficient from performance point of view, since:
  - a clock cycle time (i.e. clock rate) must be chosen such that the longest instruction can be executed in one clock cycle and
  - that makes shorter instructions execute in one unnecessarily long cycle.
- Additionally, no resource in the design may be used more than once per instruction, **thus some resources will be duplicated**.
- The single cycle design will require:
  - two memories (instruction and data),
  - two additional adders.

# Elements for Datapath Design



## Abstract /Simplified View (1<sup>st</sup> look)



- This generic implementation:
    - uses the program counter (PC) to supply instruction address,
    - gets the instruction from memory,
    - reads registers,
    - uses the instruction opcode to decide exactly what to do.
- g. babic Presentation G 5

## Abstract /Simplified View (2<sup>nd</sup> look)

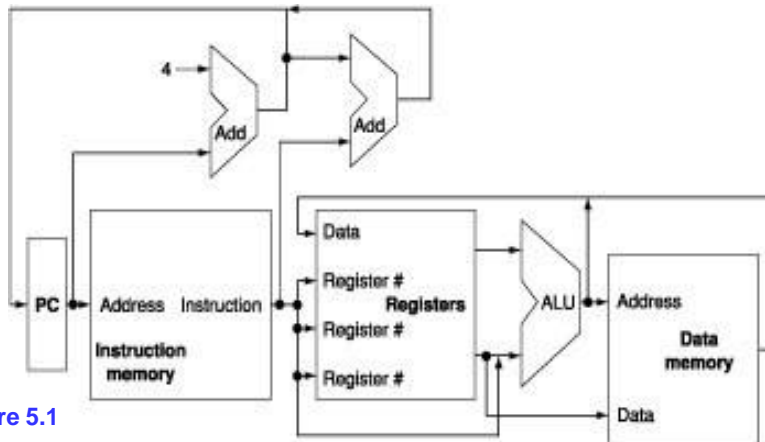


Figure 5.1

- PC is incremented by 4 by most instructions, and 4 + 4×offset by branch instructions.
- Jump instructions change PC differently (not shown).

## Our Implementation

- An edge triggered methodology
- Typical execution:
  - read contents of some state elements at the beginning of the clock cycle,
  - send values through some combinational logic,
  - write results to one or more state elements at the **end** of the clock cycle.

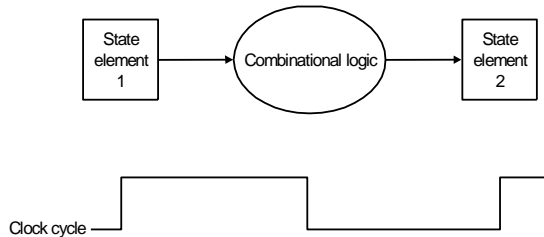


Figure 5.5

- An edge triggered methodology allows a state element to be read and written in the same clock cycle.

g. babic

Presentation G

7

## Incrementing PC & Fetching Instruction

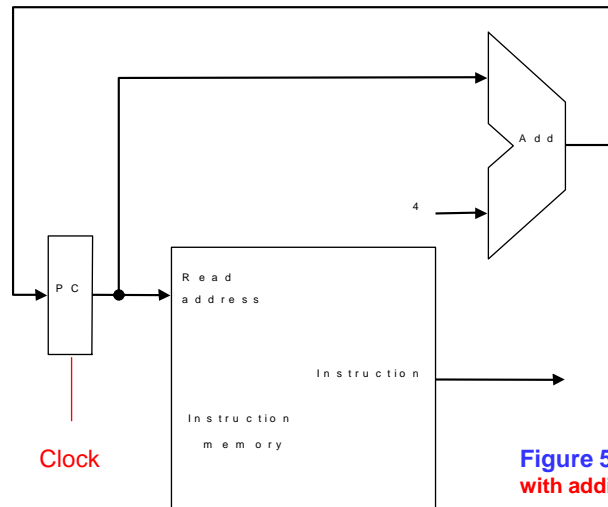


Figure 5.6  
with addition in red

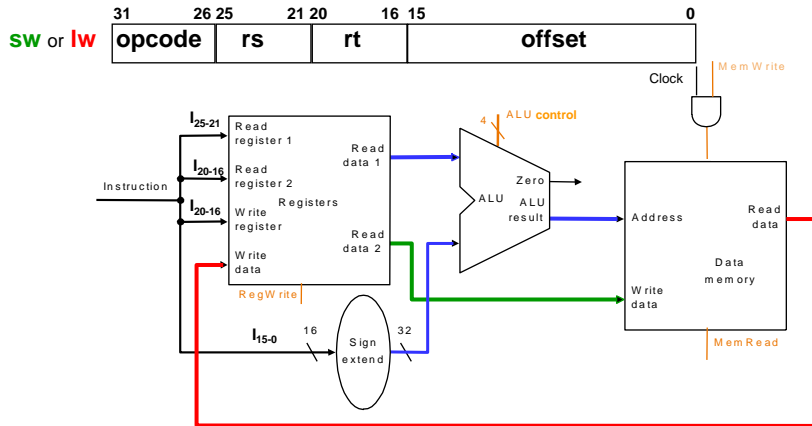
g. babic

Presentation G

8



## Datapath for LW and SW Instructions



### Control Unit sets:

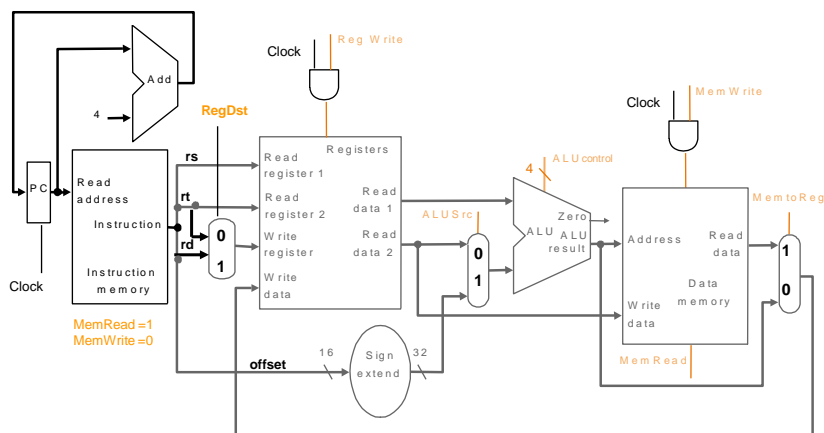
- ALU control = 0010 (add) for address calculation for both lw and sw
- MemRead=0, MemWrite=1 and RegWrite=0 for sw
- MemRead=1, MemWrite=0 and RegWrite=1 for lw

g. babic

Presentation G

11

## Datapath for R-type, LW & SW Instructions



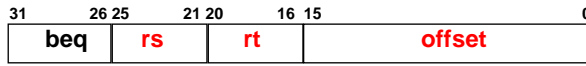
Let us determine setting of control lines for R-type, lw & sw instructions.

g. babic

Presentation G

12

# Datapath for BEQ Instruction



Branch target = [PC] + 4 + 4xoffset

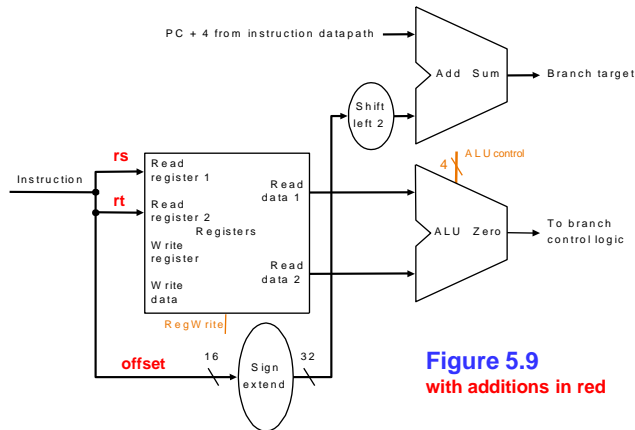


Figure 5.9 with additions in red

g. babic

Presentation G

13

# Datapath for R-type, LW, SW & BEQ

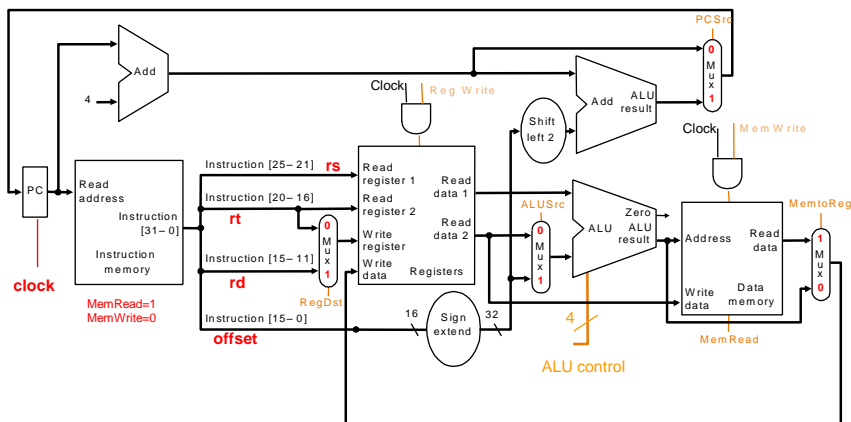


Figure 5.15 with additions in red

g. babic

Presentation G

14

# Control Unit and Datapath

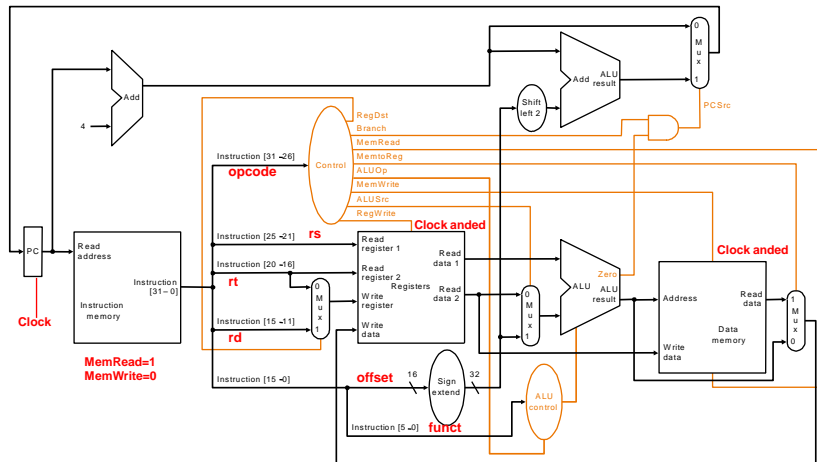


Figure 5.17  
with additions in red

g. babic

Presentation G

15

## Truth Table for (Main) Control Unit

	Input			Output						
	Op-code	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0
R-type	000000	1	0	0	1	d	0	0	1	0
lw	100011	0	1	1	1	1	0	0	0	0
sw	101011	d	1	d	0	0	1	0	0	0
beq	000100	d	0	d	0	d	0	1	0	1

- $ALUOp[1-0] = 00 \rightarrow$  signal to ALU Control unit for ALU to perform add function, i.e. set  $Ainvert = 0$ ,  $Binvert = 0$  and  $Operation = 10$
- $ALUOp[1-0] = 01 \rightarrow$  signal to ALU Control unit for ALU to perform subtract function, i.e. set  $Ainvert = 0$ ,  $Binvert = 1$  and  $Operation = 10$
- $ALUOp[1-0] = 10 \rightarrow$  signal to ALU Control unit to look at bits  $I_{[5-0]}$  and based on its pattern to set  $Ainvert$ ,  $Binvert$  and  $Operation$  so that ALU performs appropriate function, i.e. add, sub, slt, and, or & nor

g. babic

Presentation G

16

## Truth Table of ALU Control Unit

Input								Output			
ALUOp		Funct field						ALU Control			
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0				
0	0	d	d	d	d	d	d	0	0	10	add
0	1	d	d	d	d	d	d	0	1	10	sub
1	0	1	0	0	0	0	0	0	0	10	add
1	0	1	0	0	0	1	0	0	1	10	sub
1	0	1	0	0	1	0	0	0	0	00	and
1	0	1	0	0	1	0	1	0	0	01	or
1	0	1	0	1	0	1	0	0	1	11	sll
1	0	1	0	0	1	1	1	1	1	00	nor

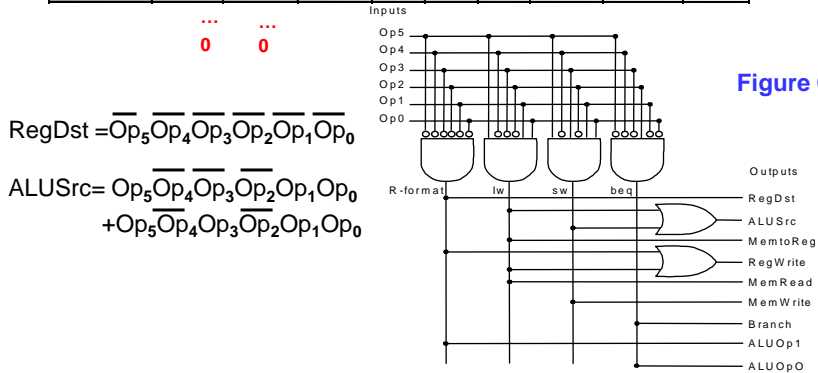
↑ Ainvert   
 ↑ Binvert   
 ↑ Operation

g. babic

17

## Design of (Main) Control Unit

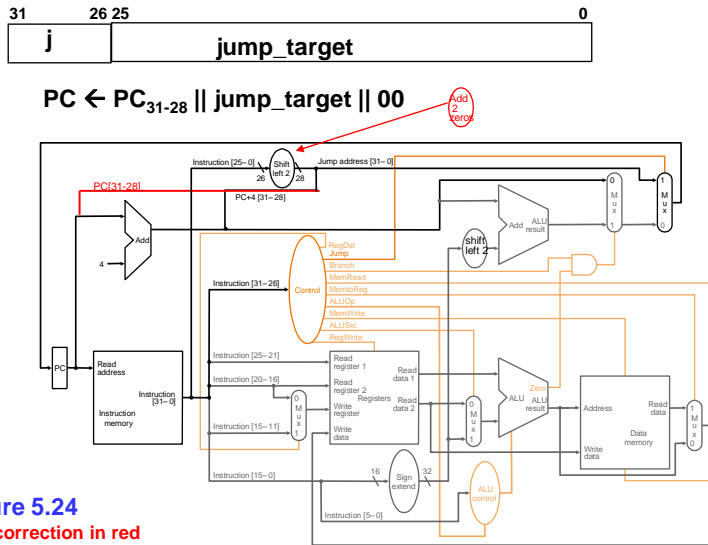
Op-code bits	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0
5 4 3 2 1 0									
0 0 0 0 0 0	1	0	0	1	d	0	0	1	0
1 0 0 0 1 1	0	1	1	1	1	0	0	0	0
1 0 1 0 1 1	d	0	1	0	0	1	0	0	0
0 0 0 1 0 0	d	0	d	0	d	0	1	0	1



g. babic

18

# Datapath for R-type, LW, SW, BEQ & J

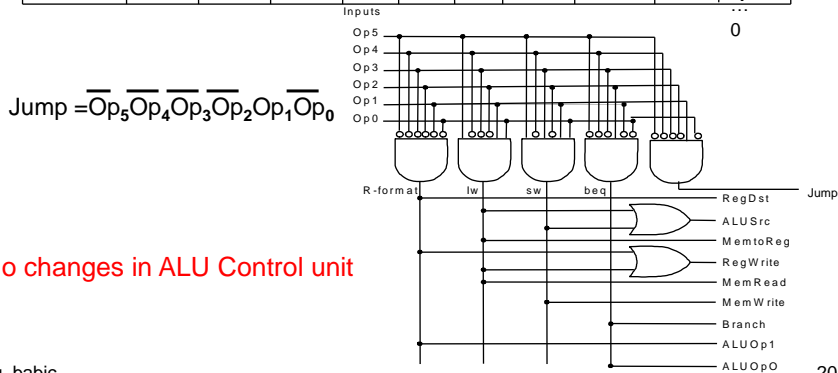


g. babic

19

# Design of Control Unit (J included)

Op-code bits	RegDst	ALUSrc	MemtoReg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0	Jump
000000	1	0	0	1	d	0	0	1	0	0
100011	0	1	1	1	1	0	0	0	0	0
101011	d	1	d	0	0	1	0	0	0	0
000100	d	0	d	0	d	0	1	0	1	0
J 000010	d	d	d	0	d	0	d	d	d	1



g. babic

20

## Cycle Time Calculation

---

- Let us assume that the **only** delays introduced are by the following tasks:
  - Memory access (read and write time = 3 nsec)
  - Register file access (read and write time = 1 nsec)
  - ALU to perform function (= 2 nsec)
- Under those assumption here are instruction execution times:

	Instr fetch		Reg read		ALU oper		Data memory		Reg write	Total
R-type	3	+	1	+	2	+			1	= 7 nsec
lw	3	+	1	+	2	+	3	+	1	= 10 nsec
sw	3	+	1	+	2	+	3			= 9 nsec
branch	3	+	1	+	2					= 6 nsec
jump	3									= 3 nsec

- Thus a clock cycle time has to be 10nsec, and clock rate = 1/10 nsec = 100MHz

## Single Cycle Processor: Conclusion

---

- Single Cycle Problems:
  - what if we had a more complicated instruction like floating point?
  - a clock cycle would be much longer,
  - thus for shorter and more often used instructions, such as add & lw, wasteful of time.
- One Solution:
  - use a "smaller" cycle time, and
  - have different instructions take different numbers of cycles.
- And that is a "multi-cycle" processor.**