



University of
Nottingham

UK | CHINA | MALAYSIA

Single-Cycle CPU

Dr. Heng Yu

AY2023-24, Spring Semester
COMP1047: Systems and Architecture
Week 5

Today's outline

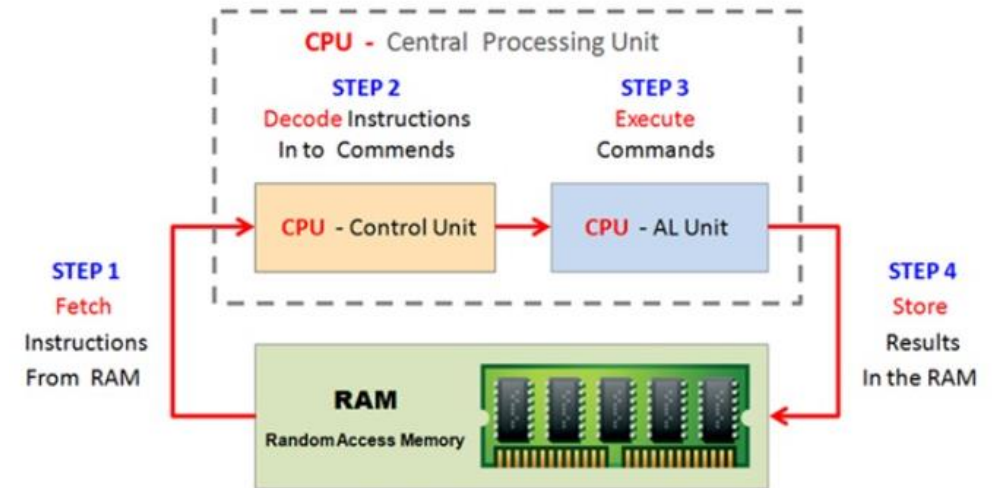
- ✓ Introducing the MIPS CPU design
- ✓ Analyzing the instruction set
- ✓ Constructing the datapath
- ✓ Datapath with control unit



- ✓ CPU performance factors
 - Instruction count, determined by ISA and compiler
 - CPI and cycle time, determined by CPU hardware
- ✓ We will examine the most basic MIPS implementation
 - Single-Cycled CPU
- ✓ Simple subset, shows most aspects
 - Memory reference: `lw`, `sw`
 - Arithmetic/logical: `add`, `sub`, `addi`, `and`, `or`, `slt`
 - Control transfer: `beq`

Instruction execution process (general)

1. PC -> instruction memory, fetch instruction
2. Register numbers -> register file, read registers
3. Depending on instruction types
 1. Use ALU to calculate
 1. Arithmetic result
 2. Memory address for load/store
 3. Branch target address
 2. Access data memory for load/store
4. PC -> target address or PC + 4



0xA000 0114

0xA000 0110

0xA000 010C

0xA000 0108

0xA000 0104

0xA000 0100

or \$10, \$2, \$7

beq \$6, \$7, L1

lw \$10, 2(\$7)

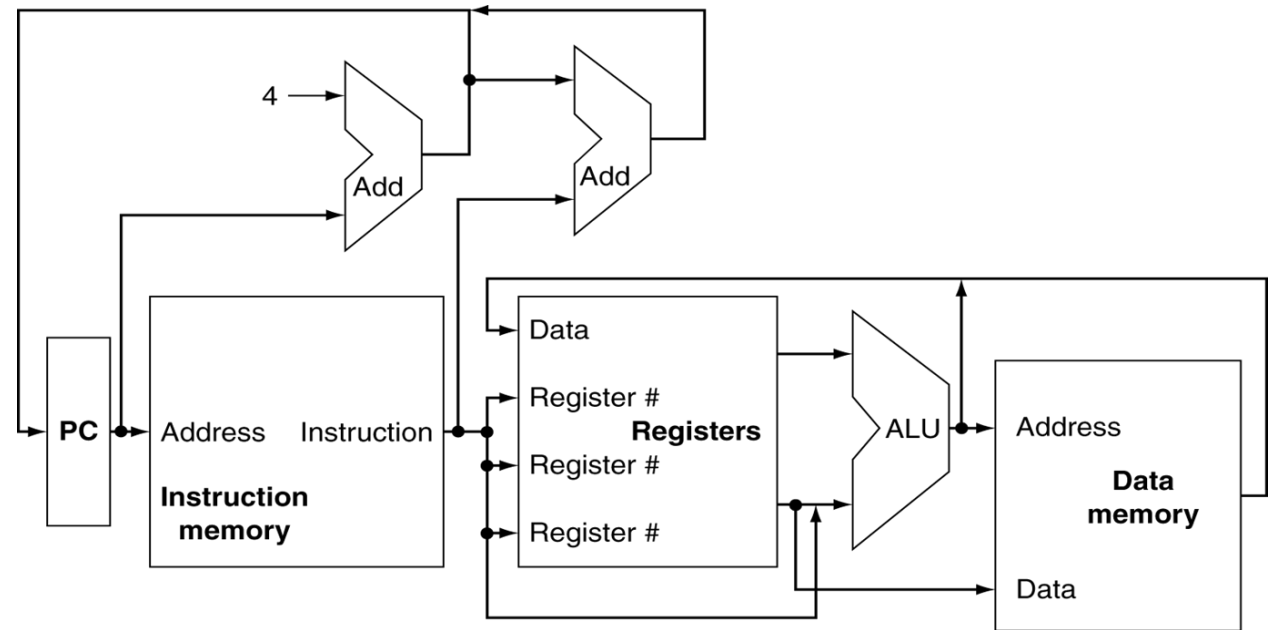
add \$6, \$7, \$8

sub \$1, \$2, \$0

add \$1, \$2, \$0

Instruction execution process (general)

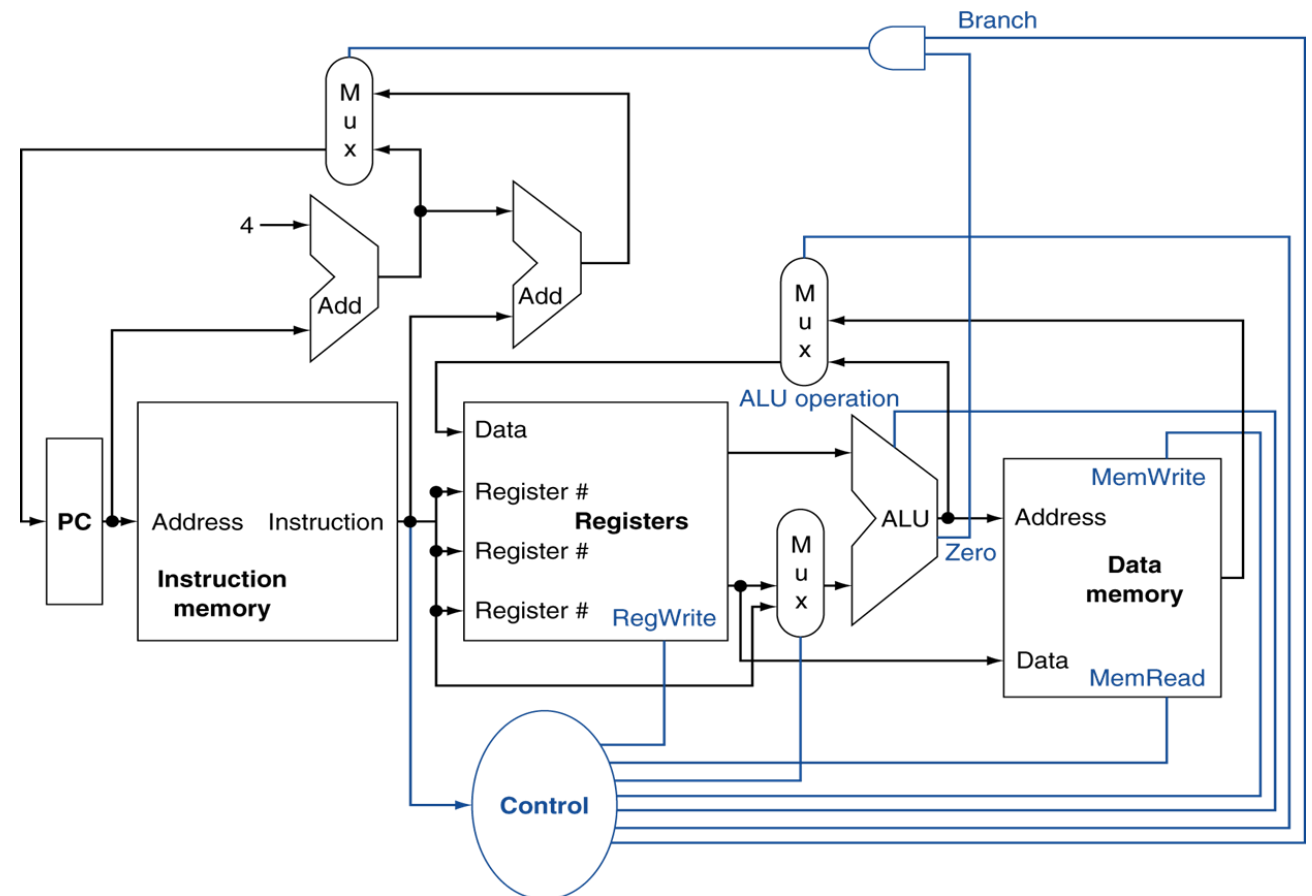
1. PC -> instruction memory, fetch instruction
2. Register numbers -> register file, read registers
3. Depending on instruction types
 1. Use ALU to calculate
 1. Arithmetic result
 2. Memory address for load/store
 3. Branch target address
 2. Access data memory for load/store
4. PC -> target address or PC + 4





Instruction execution process (general)

1. PC -> instruction memory, fetch instruction
2. Register numbers -> register file, read registers
3. Depending on instruction types
 1. Use ALU to calculate
 1. Arithmetic result
 2. Memory address for load/store
 3. Branch target address
 2. Access data memory for load/store
4. PC -> target address or PC + 4

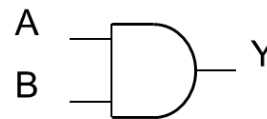


Logic Design Basics

- ✓ Information encoded in binary
 - Low voltage = 0, High voltage = 1
 - One wire per bit
 - Multi-bit data encoded on multi-wire buses
- ✓ Combinational element
 - Operate on data
 - Output is a function of input
- ✓ State (sequential) elements
 - Store information

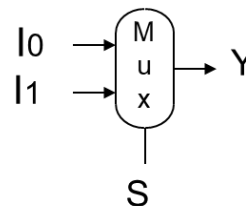
◆ AND-gate

◆ $Y = A \& B$



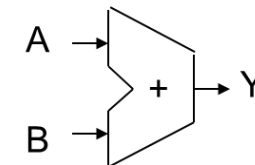
◆ Multiplexer

◆ $Y = S ? I1 : I0$



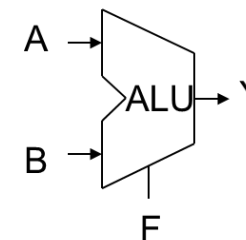
◆ Adder

◆ $Y = A + B$



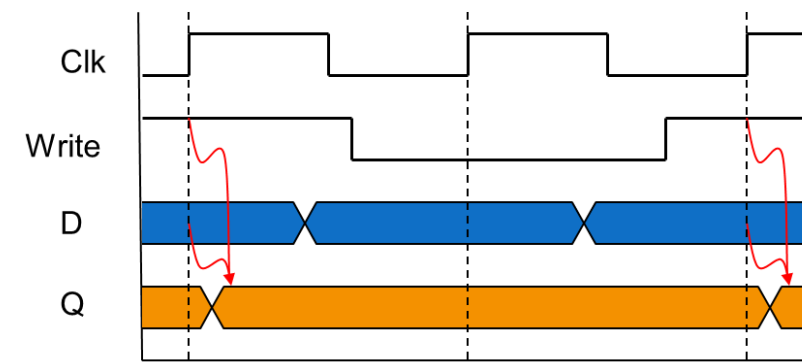
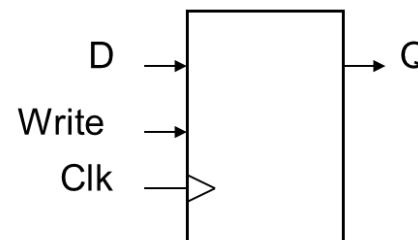
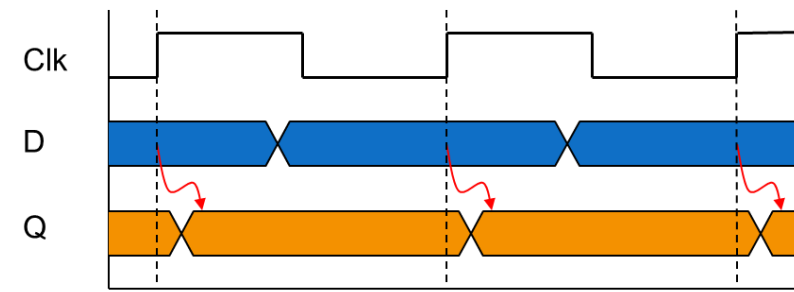
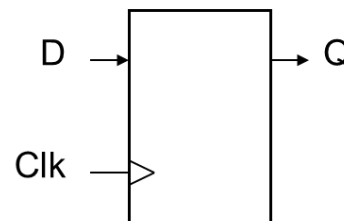
◆ Arithmetic/Logic Unit

◆ $Y = F(A, B)$



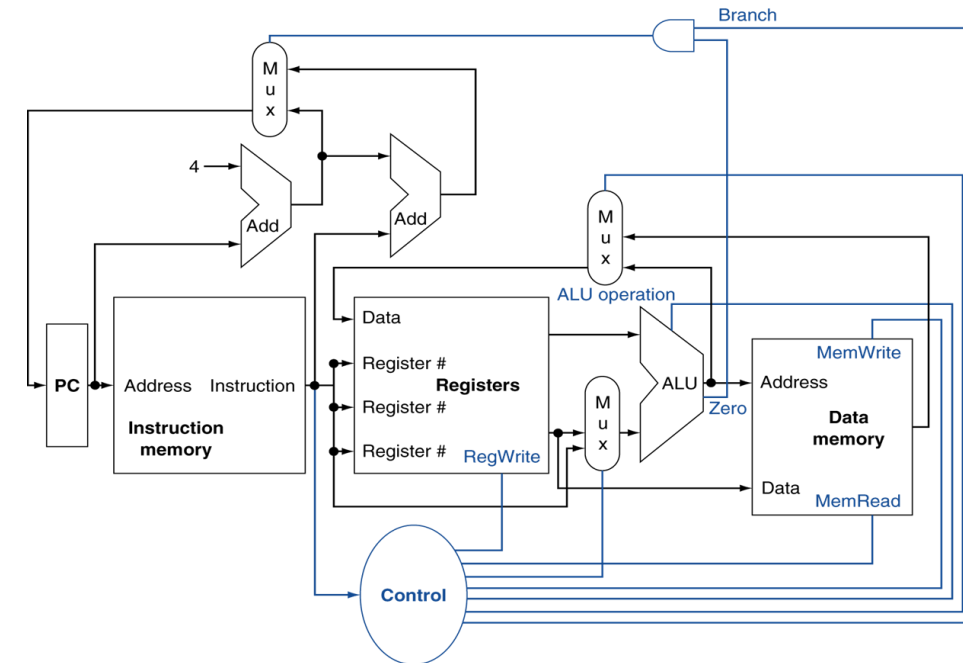
Sequential Elements

- Register: stores data in a circuit
 - Uses a clock signal to determine when to update the stored value
 - Rising-Edge-triggered**: update when Clk changes from 0 to 1
- Register with **write control**
 - Only updates on clock edge **when write control input is 1**
 - Used when stored value is required later



How to design a Processor?

1. Analyze the instruction set (datapath requirements) **(D)**
 - The meaning of each instruction is given by the *register transfers*
 - Datapath must include storage element
 - Datapath must support each register transfer
2. Select the set of datapath components and establish clocking methodology **(D)**
3. Assemble datapath meeting the requirements **(D)**
4. Analyze the implementation of each instruction to determine the setting of control points that affect the register transfer **(C)**
5. Assemble the control logic **(C)**



add \$t3, \$t1, \$t2

Today's outline

- ✓ Introducing the MIPS CPU design
- ✓ Analyzing the instruction set
- ✓ Constructing the datapath
- ✓ Datapath with control unit



Analyzing the Instruction Set

Our example: A MIPS Subset

✓ R-Type:

- ★ `add rd, rs, rt`
- ★ `sub rd, rs, rt`
- ★ `and rd, rs, rt`
- ★ `or rd, rs, rt`
- ★ `slt rd, rs, rt`

✓ Load/Store:

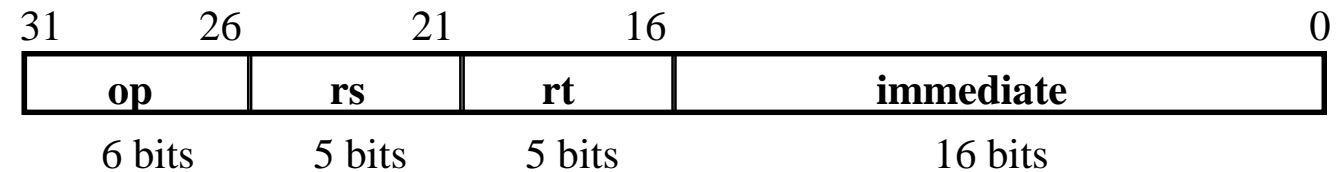
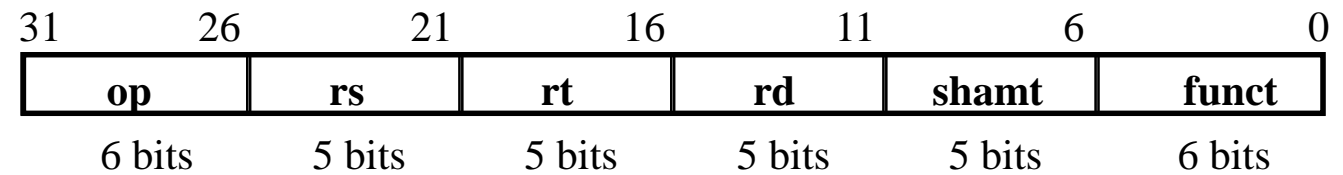
- ★ `lw rt,rs,imm16`
- ★ `sw rt,rs,imm16`

✓ Imm operand:

- ★ `addi rt,rs,imm16`

✓ Branch:

- ★ `beq rs,rt,imm16`





Analyzing the Instruction Set

RTL (Register Transfer Language) gives the semantics of the instructions

$MEM[PC] = op \mid rs \mid rt \mid rd \mid shamt \mid funct$
or $= op \mid rs \mid rt \mid Imm16$

Inst.	Register transfers
ADD	$R[rd] \leftarrow R[rs] + R[rt]; \quad PC \leftarrow PC + 4$
SUB	$R[rd] \leftarrow R[rs] - R[rt]; \quad PC \leftarrow PC + 4$
LOAD	$R[rt] \leftarrow MEM[R[rs] + sign_ext(Imm16)]; \quad PC \leftarrow PC + 4$
STORE	$MEM[R[rs] + sign_ext(Imm16)] \leftarrow R[rt]; \quad PC \leftarrow PC + 4$
ADDI	$R[rt] \leftarrow R[rs] + sign_ext(Imm16); \quad PC \leftarrow PC + 4$
BEQ	if $(R[rs] == R[rt])$ then $PC \leftarrow PC + 4 + sign_ext(Imm16) \mid \mid 00$ else $PC \leftarrow PC + 4$

0xA000 0114

0xA000 0110

0xA000 010C

0xA000 0108

0xA000 0104

0xA000 0100

or \$10, \$2, \$7

beq \$6, \$7, L1

lw \$10, 2(\$7)

add \$6, \$7, \$8

sub \$1, \$2, \$0

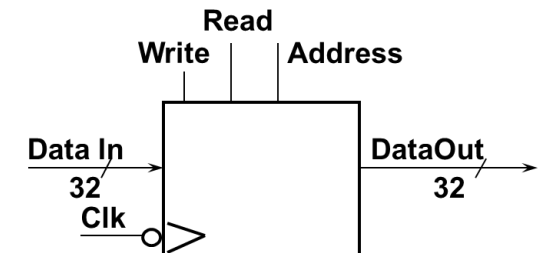
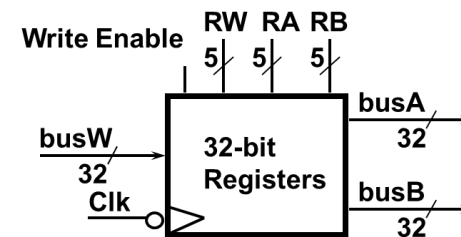
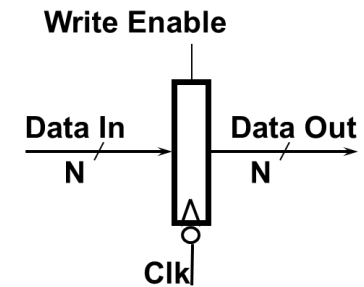
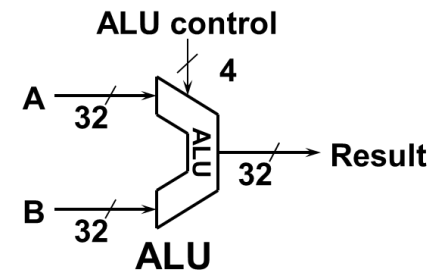
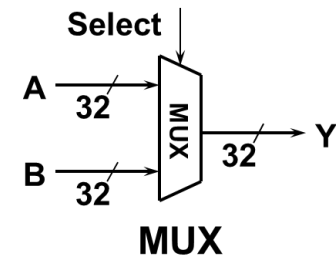
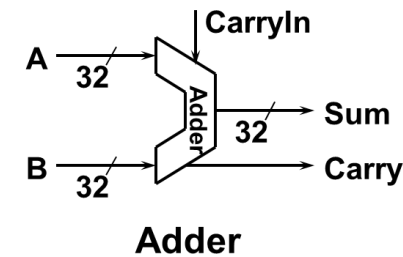
add \$1, \$2, \$0

Analyzing the Instruction Set

What hardware are needed?

After checking the register transfers, we can conclude that datapath needs the following hardware:

- ➔ Memory
 - store instructions and data
- ➔ Registers (32 x 32)
 - read RS
 - read RT
 - Write RT or RD
- ➔ PC
- ➔ Extender for zero- or sign-extension
- ➔ Calculating values in registers or extended immediate (ALU)
- ➔ Add 4 or extended immediate to PC



Today's outline

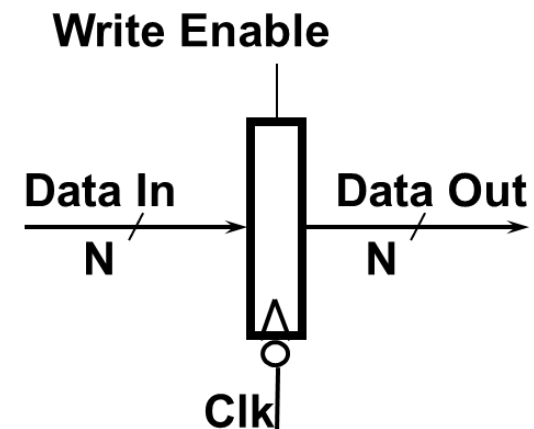
- ✓ Introducing the MIPS CPU design
- ✓ Analyzing the instruction set
- ✓ Constructing the datapath
- ✓ Datapath with control unit



Sequential components for datapath:

Register:

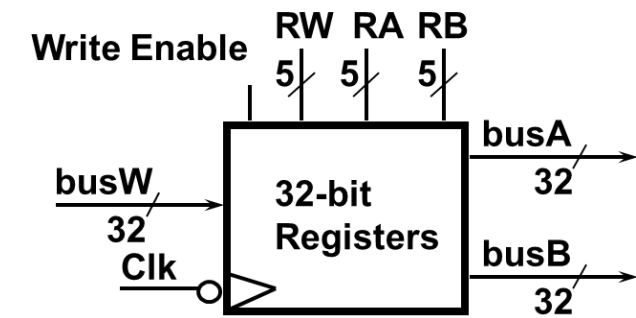
- ✓ Similar to the D Flip Flop except
 - 32-bit input and output
 - Write Enable input
- ✓ Write Enable:
 - Negated (0): **Data Out** will not change
 - Asserted (1): **Data Out** will become **Data In**



Sequential components for datapath:

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** selects the register to put on **busA** (data)
 - **RB** selects the register to put on **busB** (data)
 - **RW** selects the register to be written via **busW** (data) when **Write Enable** is 1
- ✓ Clock input (CLK)
 - The **CLK** input is a factor ONLY during write operation
 - During read, behaves as a combinational circuit

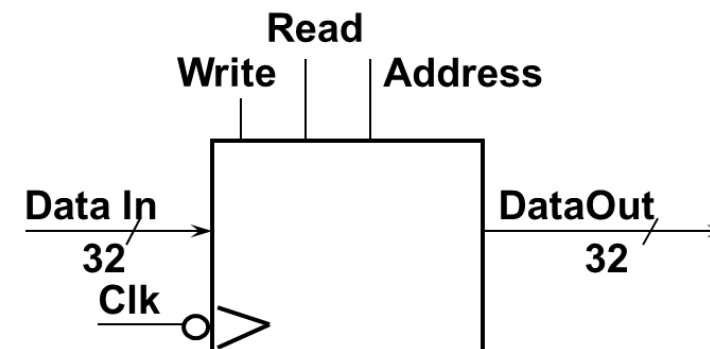




Sequential components for datapath:

Memory:

- ✓ One input bus: **Data In**
- ✓ One output bus: **Data Out**
- ✓ Word is selected 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)
 - The **CLK** input is a factor during write and read operations



0xA000 0114
0xA000 0110
0xA000 010C
0xA000 0108
0xA000 0104
0xA000 0100

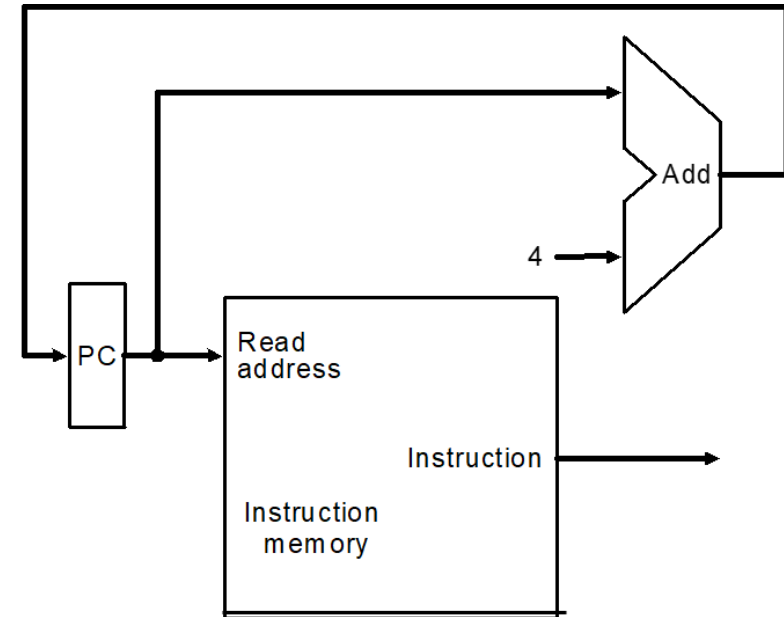
or \$10, \$2, \$7
beq \$6, \$7, L1
lw \$10, 2(\$7)
add \$6, \$7, \$8
sub \$1, \$2, \$0
add \$1, \$2, \$0

Assembling the datapath

Instruction fetching is divided into two major steps:

- ✓ Fetch the instruction: **mem[PC]**
- ✓ Update the program counter:
 - Sequential code: $PC \leftarrow PC + 4$
 - Branch and Jump: $PC \leftarrow \text{"Something else"}$

$MEM[PC] = op \mid rs \mid rt \mid rd \mid shamt \mid funct$
 $or \quad = op \mid rs \mid rt \mid Imm16$



0xA000 0114

0xA000 0110

0xA000 010C

0xA000 0108

0xA000 0104

0xA000 0100

or \$10, \$2, \$7

beq \$6, \$7, L1

lw \$10, 2(\$7)

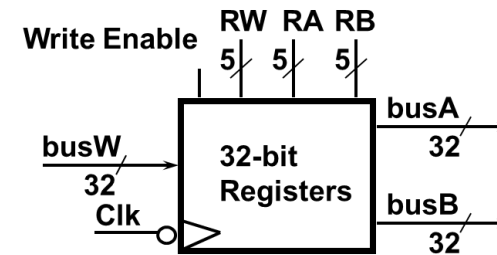
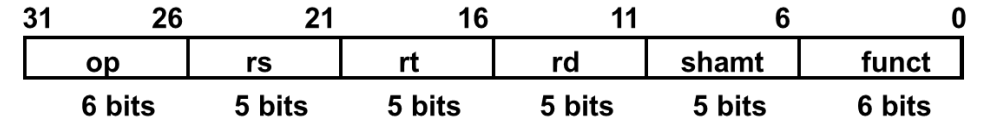
add \$6, \$7, \$8

sub \$1, \$2, \$0

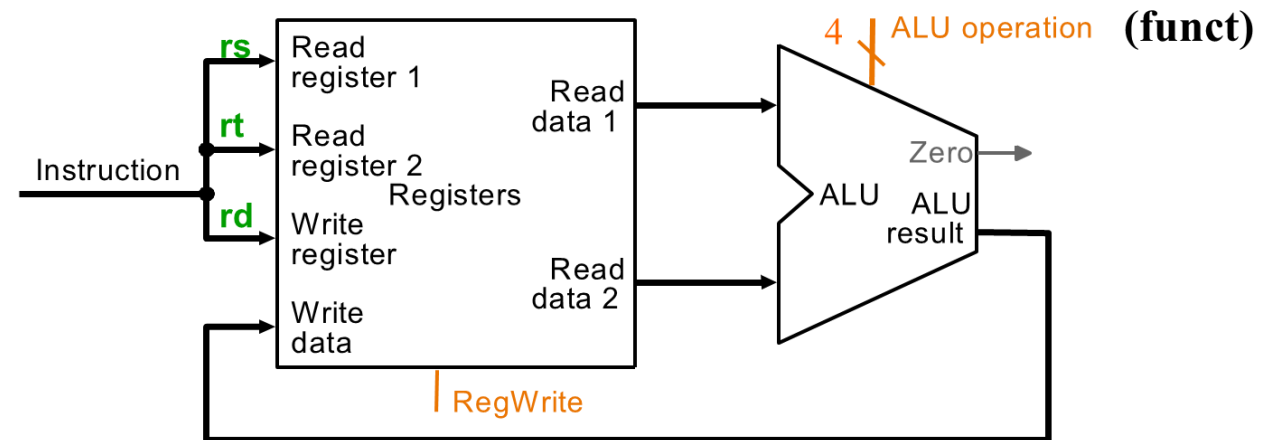
add \$1, \$2, \$0

Assembling the datapath

- ✓ **Arithmetic/logical operations:** $R[rd] \leftarrow R[rs] \text{ op } R[rt]$
- ✓ E.g., R-type instructions: add, sub, and, or, etc.
- ✓ **RA, RB, RW** come from instruction's **rs, rt, rd** fields
- ✓ ALU and RegWrite: control logic after decoding



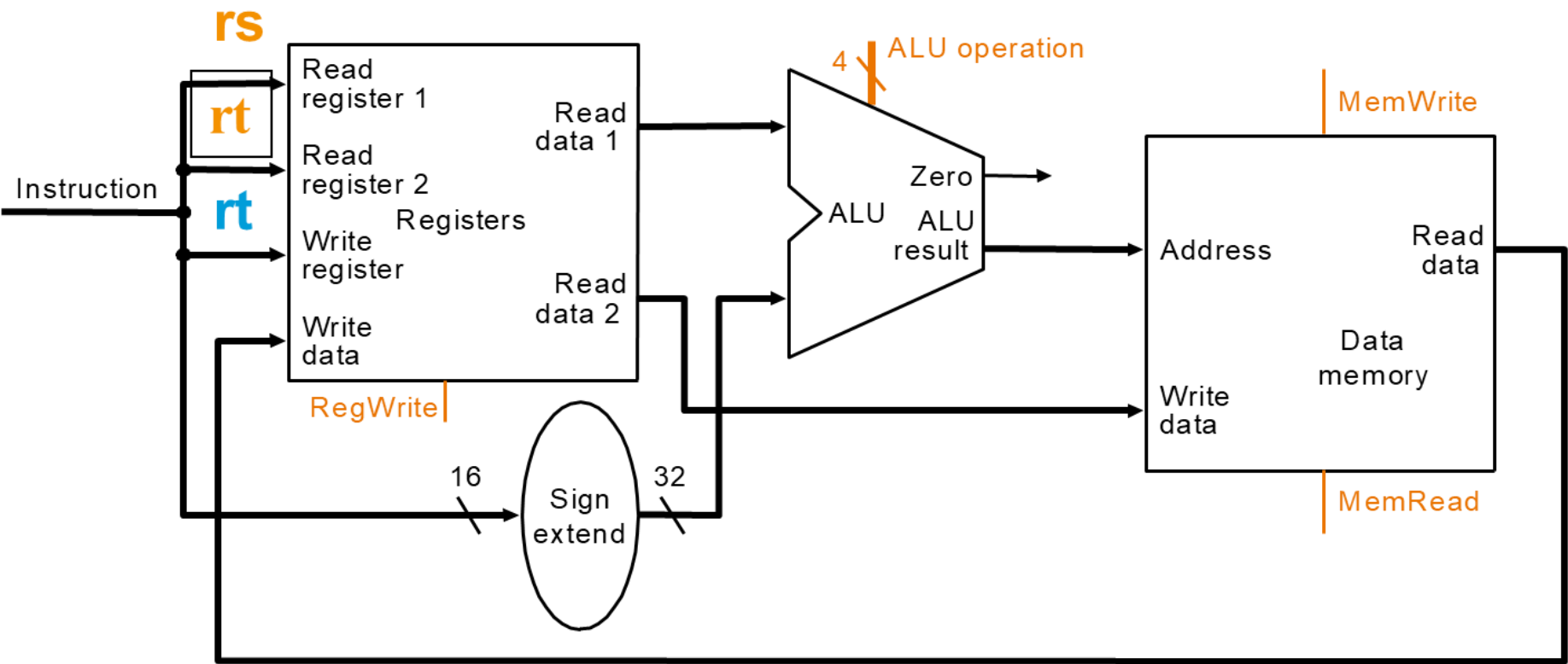
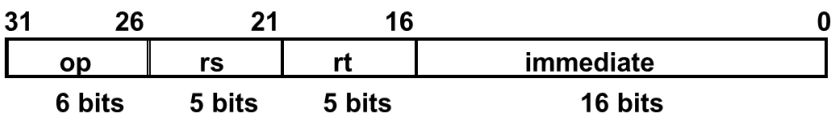
ADD $R[rd] \leftarrow R[rs] + R[rt];$ $PC \leftarrow PC + 4$
SUB $R[rd] \leftarrow R[rs] - R[rt];$ $PC \leftarrow PC + 4$



Assembling the datapath

load operations: lw rt, imm16(rs)

$R[rt] \leftarrow MEM[R[rs] + \text{sign_ext}(Imm16)];$



lw \$10, 4(\$9)

0xA000 0108
0xA000 0104
0xA000 0100

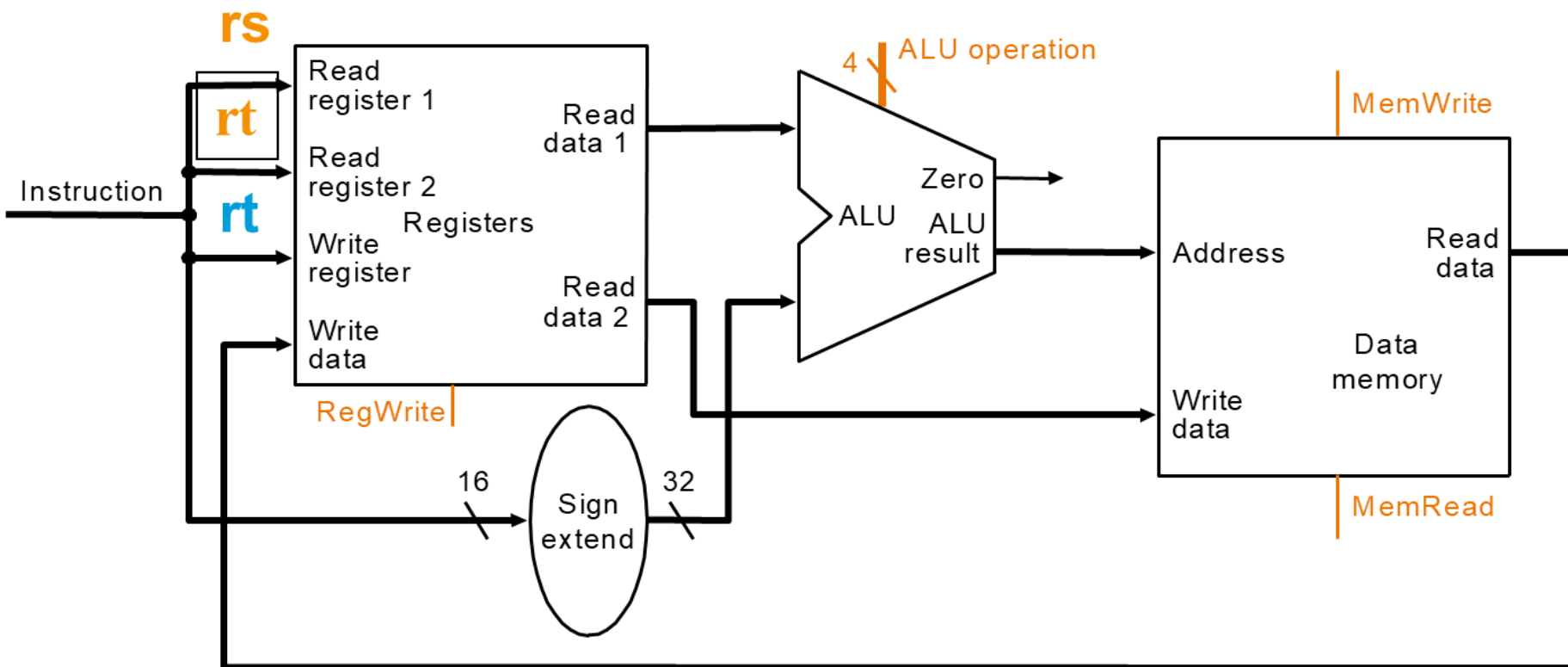
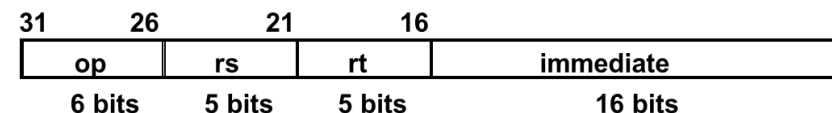
xxxx ... xxxx
0100 ... 1001
0000 ... 0100



Assembling the datapath

✓ store operations: **sw** **rt**, **imm16(rs)**

$\text{MEM}[\text{R}[\text{rs}] + \text{sign_ext}(\text{Imm16})] \leftarrow \text{R}[\text{rt}]$



sw **\$10**, **4(\$9)**

0xA000 0108

xxxx ... xxxx

0xA000 0104

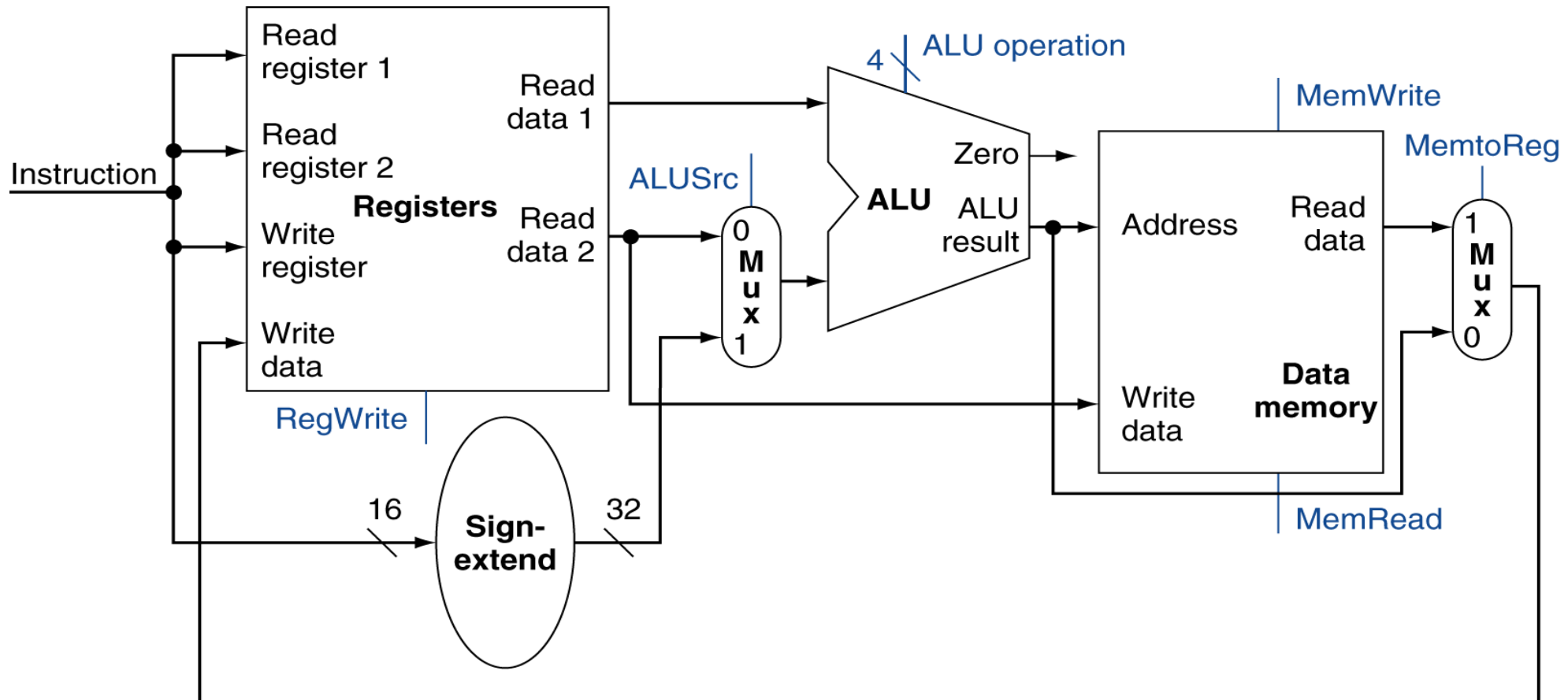
0100 ... 1001

0xA000 0100

0000 ... 0100

Assembling the datapath

✓ R-Type/Load/Store Datapath



Assembling the datapath

✓ **Branch operations:** `beq rs, rt, imm16`

`mem[PC]`

`Equal ← R[rs] == R[rt]`

`if (Equal)`

`PC ← PC + 4 + (SignExt(imm16) x 4)`

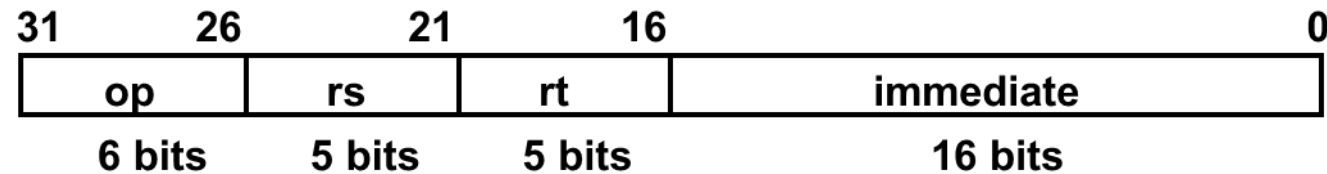
`else`

`PC ← PC + 4`

Fetch inst. from memory

Calculate branch condition

Calculate next inst. Address



Assembling the datapath

✓ **Branch operations:** `beq rs, rt, imm16`

`mem[PC]`

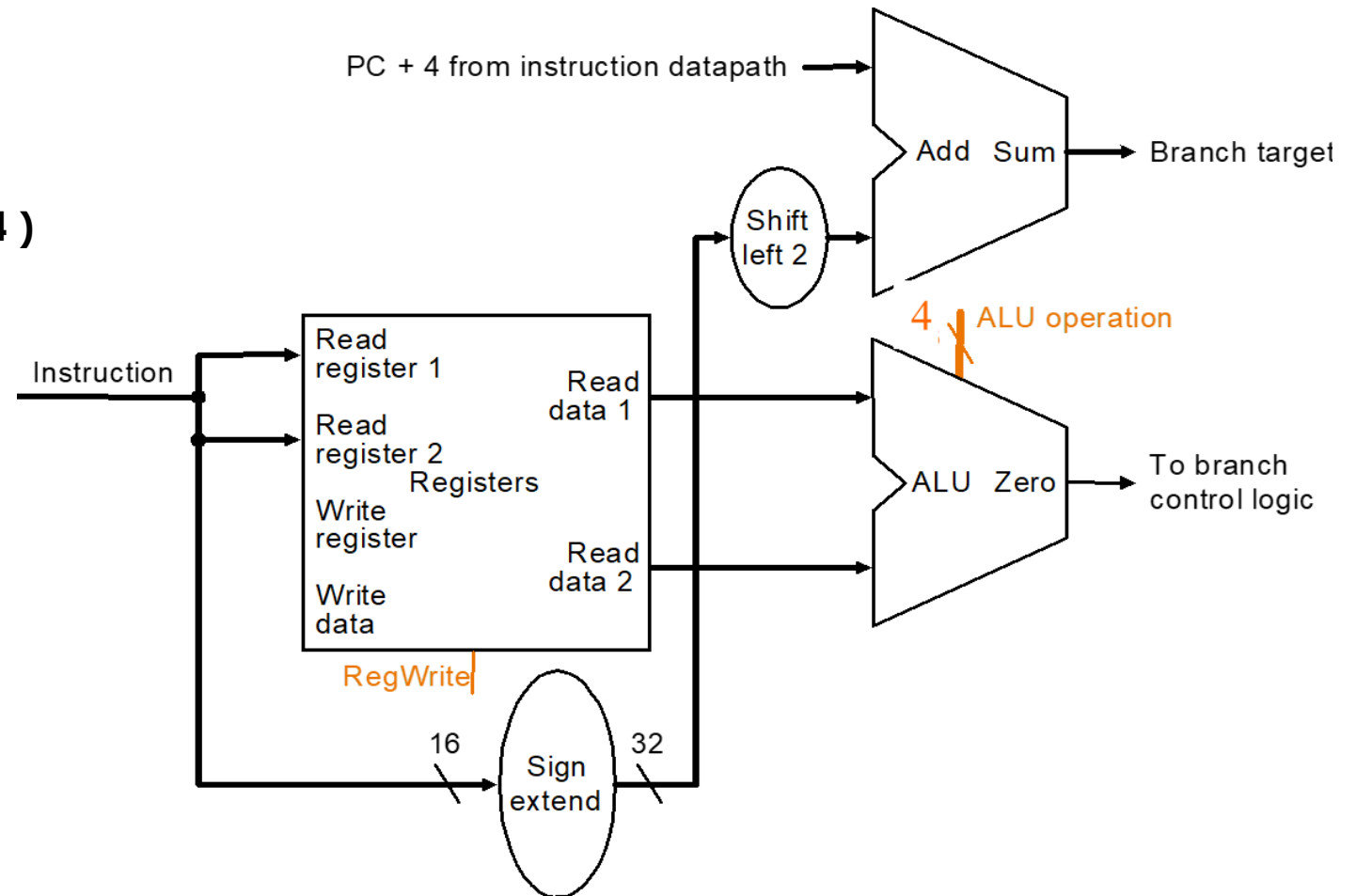
`Equal \leftarrow R[rs] == R[rt]`

`if (Equal)`

`PC \leftarrow PC + 4 + (SignExt(imm16) x 4)`

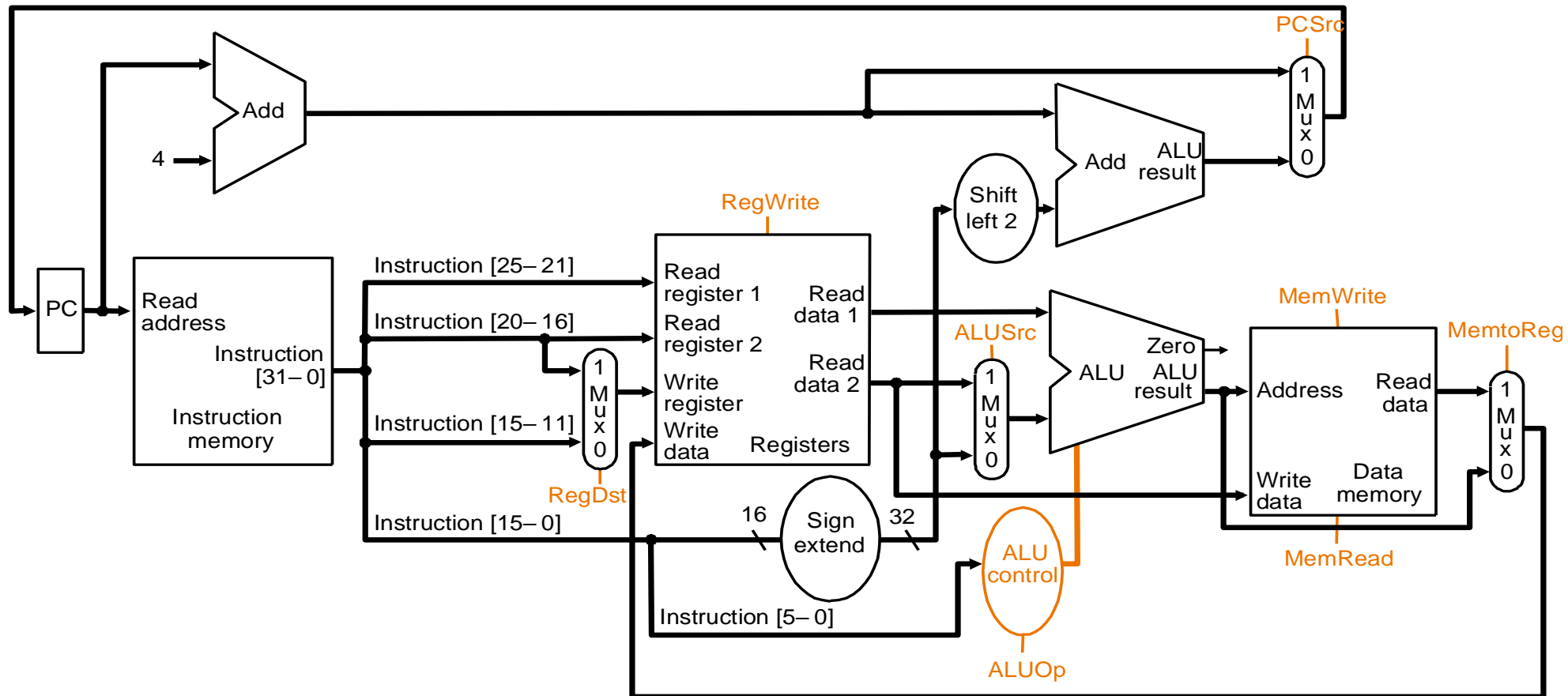
`else`

`PC \leftarrow PC + 4`





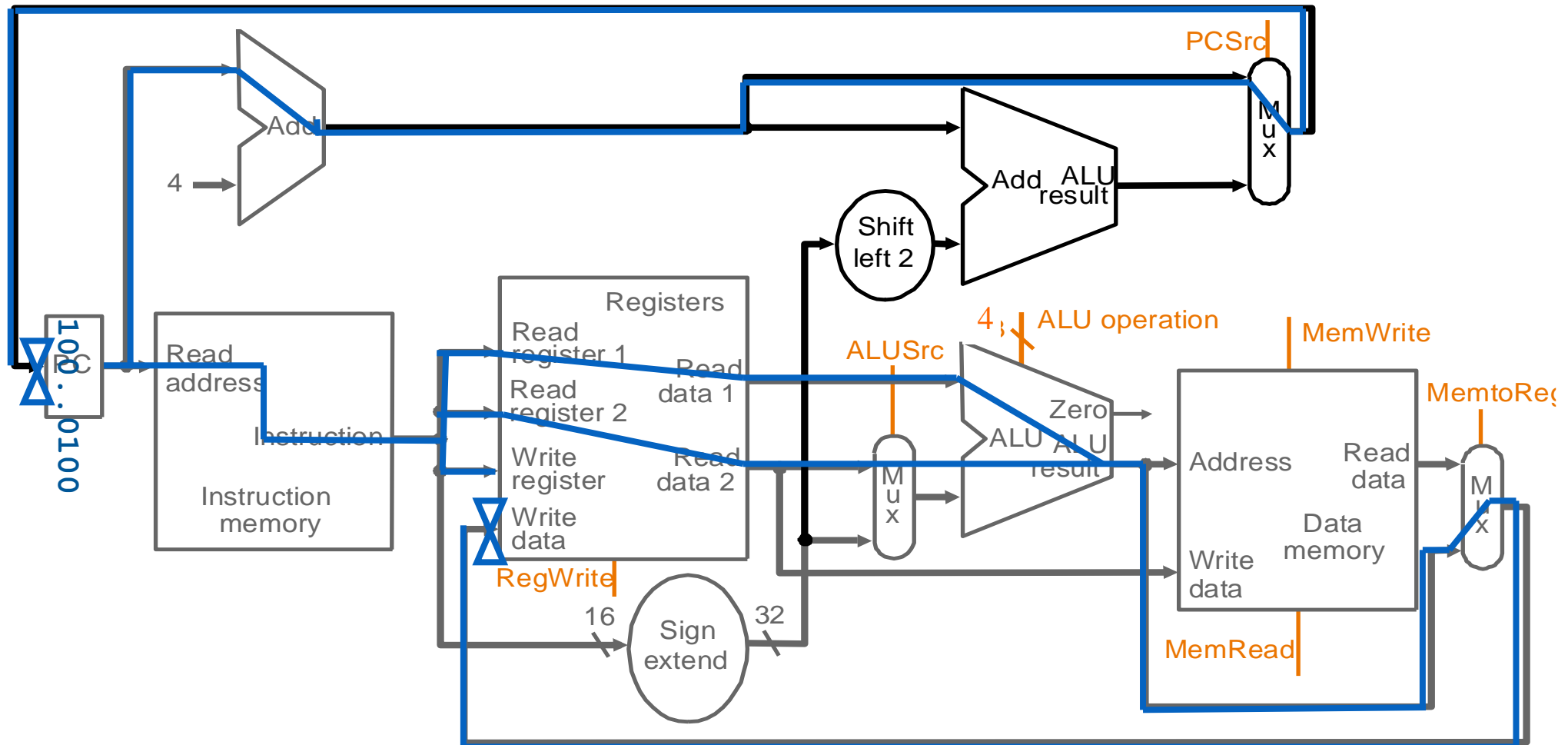
A Single-Cycle Datapath





Building the Datapath

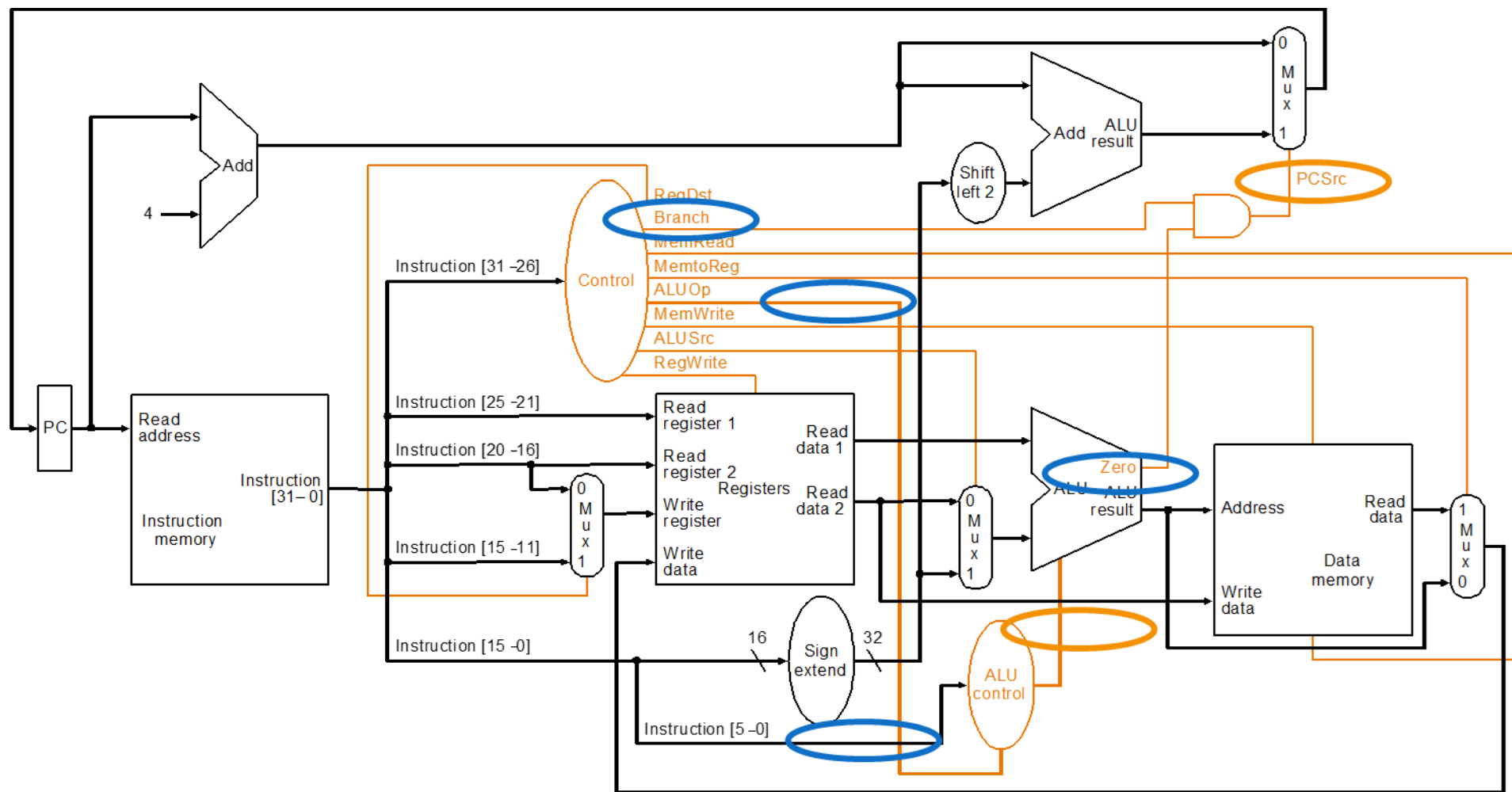
Dataflow during **add**



Today's outline

- ✓ Introducing the MIPS CPU design
- ✓ Analyzing the instruction set
- ✓ Constructing the datapath
- ✓ Datapath with control unit

Datapath with control unit

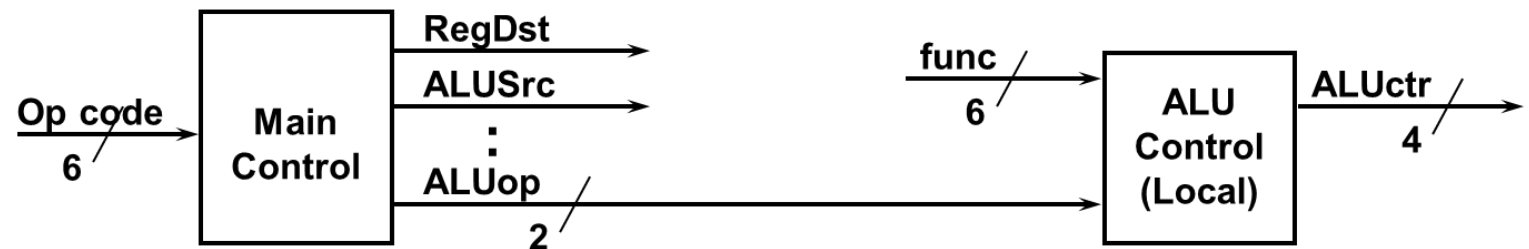




Truth Table of Control Signals (6 Inputs and 9 Outputs)

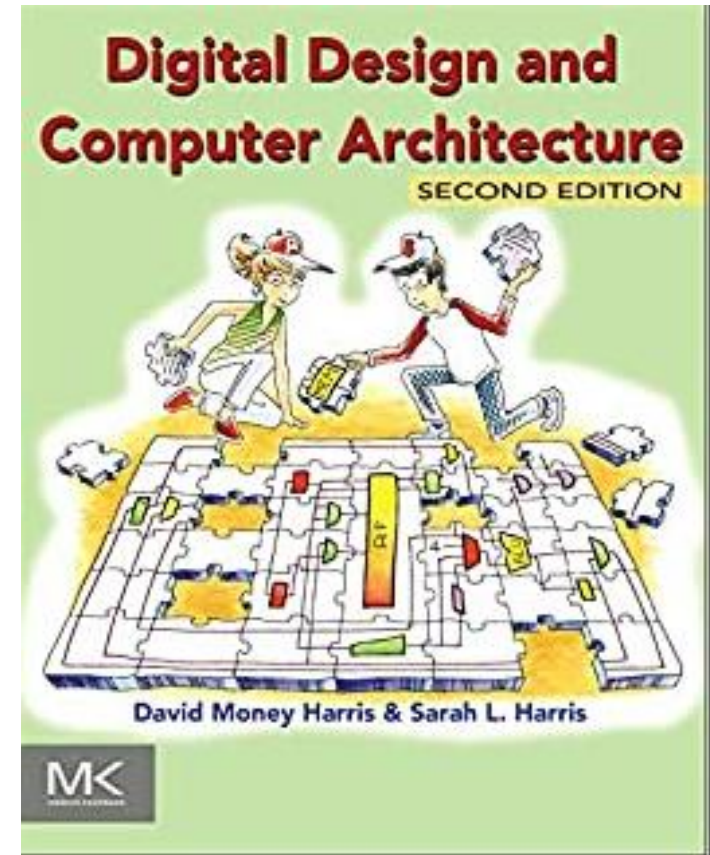
See Appendix A

	func	op	10 0000	10 0010	We Don't Care :-)		
			00 0000	00 0000	10 0011	10 1011	00 0100
			add	sub	lw	sw	beq
RegDst			1	1	0	x	x
ALUSrc			0	0	1	1	0
MemtoReg			0	0	1	x	x
RegWrite			1	1	1	0	0
MemRead			0	0	1	0	0
MemWrite			0	0	0	1	0
Branch			0	0	0	0	1
ALUop1			1	1	0	0	0
ALUop0			0	0	0	0	1



Summary

- Single cycle datapath => $CPI=1$, clock cycle time long
- MIPS makes control easier
 - Instructions in the same size
 - Source registers always in the same place
 - Immediates in the same size and location
 - Operations always on registers or immediates
- Not covered in this lecture
 - Designing the control unit
 - J-type instruction
 - Read the corresponding chapters in textbooks for your own curiosity.





University of
Nottingham

UK | CHINA | MALAYSIA

Thank you.