



The Islamic University of Gaza
Faculty of Engineering
Department of Computer Engineering



LCOM 3010 - Computer Architecture

Lab 7

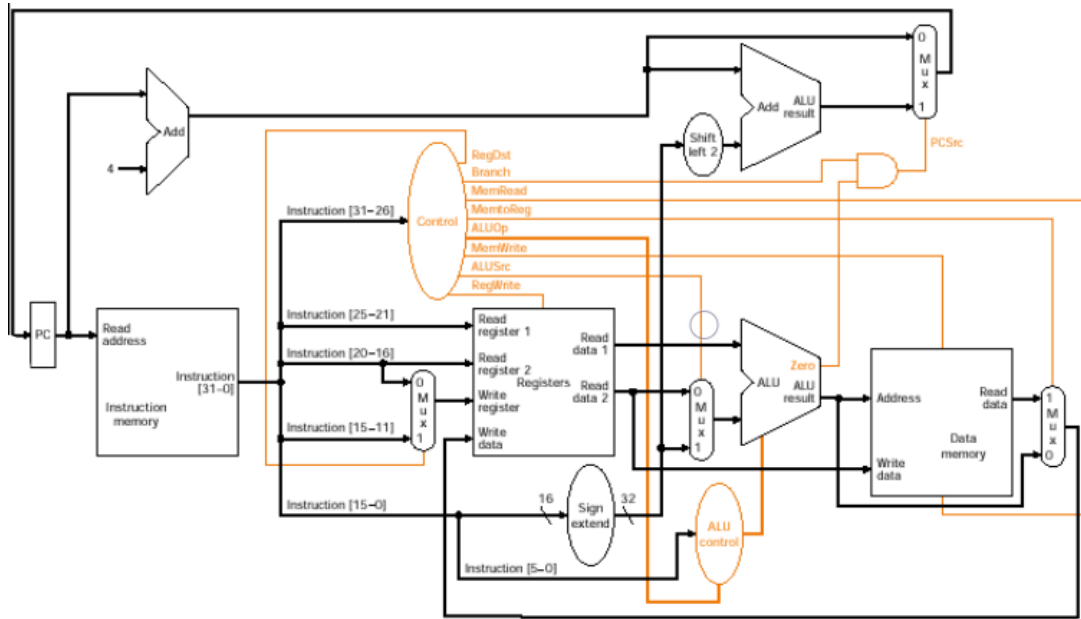
CONTROL UNIT DESIGN

Prof. Wesam M. Ashour

Eng. Hashem M. Hejazy

Summer 2022/2023

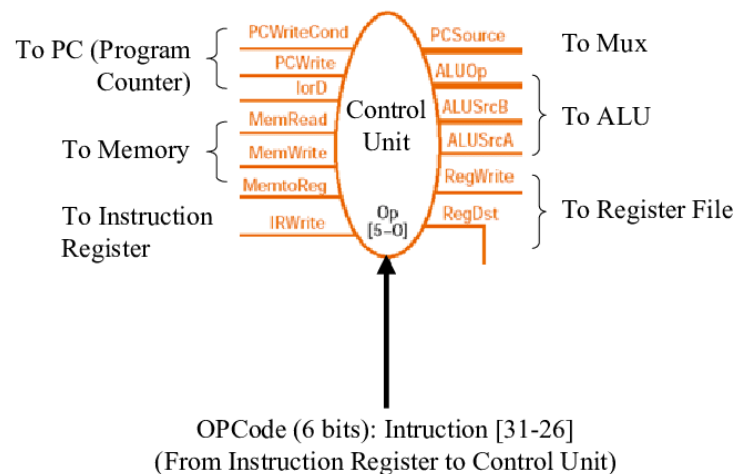
Introduction



Design of computer architecture

What is the control unit?

Control Unit is the part of the computer's central processing unit (CPU), which directs the operation of the processor. It is the responsibility of the Control Unit to tell the computer's memory, arithmetic/logic unit and input and output devices how to respond to the instructions that have been sent to the processor. It fetches internal instructions of the programs from the main memory to the processor instruction register, and based on this register contents, the control unit generates a control signal that supervises the execution of these instructions. A control unit works by receiving input information to which it converts into control signals, which are then sent to the central processor. The computer's processor then tells the attached hardware what operations to perform.



The control unit is responsible for taking the instruction and generating the appropriate signals for the datapath elements.

Signals that need to be generated include

- Operation to be performed by ALU.
- Whether register file needs to be written.
- Signals for multiple intermediate multiplexors.
- Whether data memory needs to be written.

For the most part, we can generate these signals using only the opcode and funct fields of an instruction.

Here is a table that show the value of opcode for MIPS instructions format

Format	Opcode
R-format	000000
lw	100011
sw	101011
beq	000100
j	000010

MIPS Control Signal Summary

The single-cycle implementation of the MIPS processor uses the following control signals, which are grouped according to the execution activity that they affect.

Activity	Signal	Purpose
PC Update	Branch	Combined with a condition test Boolean to enable loading the branch target address into the PC.
	Jump	Enables loading the jump target address into the PC
Source Operand Fetch	ALUSrc	Selects the second source operand for the ALU (rt or sign-extended immediate field in Patterson and Hennessey).
ALU Operation	ALUOp	Either specifies the ALU operation to be performed or specifies that the operation should be determined from the function bits.
Memory Access	MemRead	Enables a memory read for load instructions.
	MemWrite	Enables a memory write for store instructions.
Register Write	RegWrite	Enables a write to one of the registers.

	RegDst	Determines how the destination register is specified (rt or rd in Patterson and Hennessey).
	MemtoReg	Determines where the value to be written comes from (ALU result or memory in Patterson and Hennessey).

From the previous table, we can see that the control signals are chosen based on the upper 6 bits of the instruction. That is, the opcode is used to set the control lines.

Instr.	RegDst	ALUSrc	MemToReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0	jump
R	1	0	0	1	0	0	0	1	0	0
lw	0	1	1	1	1	0	0	0	0	0
sw	x	1	x	0	0	1	0	0	0	0
beq	x	0	x	0	0	0	1	0	1	0
j	x	x	x	0	0	0	0	x	x	1

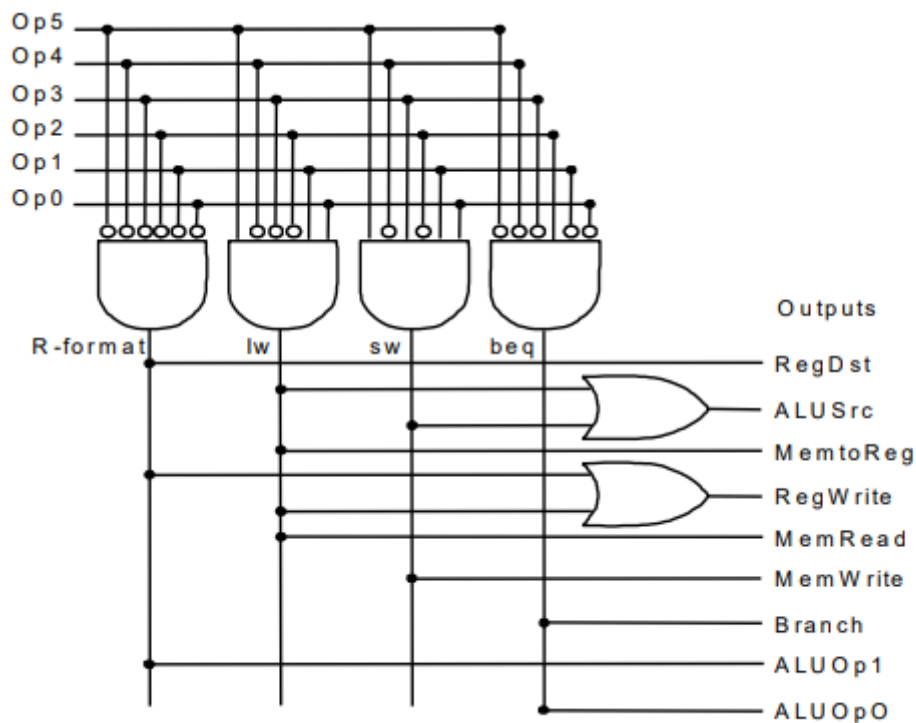
ALUOp can be encoded using 2 bits as follows

ALUOp1	ALUOp0	Instruction
0	0	Memory operation(load, store)
0	1	beq
1	0	R-type

To design the control unit you can get it from the opcode bits by get the function where the output is equal 1. For example:

RegDst =Op5'Op4'Op3'Op2'Op1'Op0'

The Design of Control Unit is as follow



ALU Control

In a MIPS processor, the ALU control bits are set based on the ALUOp control bits and the different function codes for the R-type instruction. The ALUOp control bits specify the type of operation to be performed by the ALU, while the function code field of the R-type instruction specifies the specific operation to be performed. Here is how the ALU control bits are set for different values of the ALUOp control bits and function codes:

Instruction	ALUOp	Funct	Alu operation	Alu control Input
lw	00	xxxxxx	Add	010
sw	00	xxxxxx	Add	010
beq	01	xxxxxx	Subtract	110
add	10	100000	Add	010
sub	10	100010	Subtract	110
and	10	100100	And	000
or	10	100101	Or	001
slt	10	101010	slt	111

Note: Alu control Inputs are the selection line for Alu which described in previous lab, first two bits are the selection for MUX to select the operation, and last bit is Binvs. Which used to apply the subtract operation in ALU.

Extending instruction set

Extending the instruction set in MIPS (Microprocessor without Interlocked Pipeline Stages) involves adding new instructions to the existing set to enhance the capabilities of the processor. Here's a general process for extending the MIPS instruction set:

1. **Identify the need:** Determine the specific functionality or operation that you want to add to the MIPS architecture. This could be a new arithmetic operation, a specialized instruction for a particular task, or any other feature that would benefit the processor.
2. **Define the new instruction:** Decide on the format, opcode, and other necessary fields for the new instruction. Consider the existing MIPS instruction formats (R, I, J) and their corresponding fields (op, rs, rt, rd, shamt, funct, immediate, target) to ensure compatibility with the existing architecture.
3. **Assign opcode and function code:** Select a unique opcode (6-bit field) for the new instruction. If you're creating an R-type instruction, you'll also need to assign a unique function code (6-bit field) for the instruction.
4. **Design the instruction semantics:** Specify the operation or behavior of the new instruction. Determine the source and destination registers, immediate values, or other data involved in the instruction. Consider the impact on the processor's state and any necessary side effects.
5. **Implement the instruction in hardware or software:** Depending on your goals and resources, you can implement the new instruction either in hardware (e.g., by modifying the processor's datapath and control unit) or in software (e.g., by introducing a new opcode and handling it in an assembler or compiler).
6. **Test and validate:** Develop test cases to verify the correct behavior of the new instruction. Ensure that it produces the expected results and doesn't introduce any unintended side effects or conflicts with existing instructions.

Note that extending the MIPS instruction set requires a deep understanding of computer architecture and assembly language programming. It's also important to consider the trade-offs between adding new instructions and maintaining compatibility with existing software and tools.

Add Immediate or to Instructions Set

The syntax of OR immediate is

```
ori $rt, $rs, immediate_value
```

To add an "OR immediate" instruction to the MIPS instruction set, you can follow these steps:

1. Instruction format: Determine the format for the instruction. Since it involves an immediate value, the most suitable format would be the I-type format.
2. Opcode: Choose a unique opcode for the new instruction. For example, you can assign "001101" (represented in binary) as the opcode for the "ORI" instruction.
3. Fields: Define the fields required for the instruction format. In the case of the I-type format, the fields typically include:
 - rs: The source register that holds the value to be ORed.
 - rt: The target register to which the result is stored.
 - immediate: The immediate value to be ORed with the contents of the source register.
4. Semantics: Specify the behavior of the instruction. In this case, the ORI instruction performs a bitwise OR operation between the value in the source register (rs) and the immediate value. The result is stored in the target register (rt).
5. Encoding: Combine the opcode and field values to create the binary encoding for the instruction. For example, if we choose rs = \$s1, rt = \$t1, and the immediate value is 10, the binary encoding for the ORI instruction would be:

```
opcode(6bits):001101
rs(5bits):register_s1
rt(5bits):register_t1
immediate (16 bits): 0000000000001010
```

6. Implement and test: Depending on your goals, you can implement the ORI instruction in hardware by modifying the MIPS processor's datapath and control unit, or you can handle it in software by integrating it into the assembler or compiler.

Note: The immediate value is a 16 bit so, it must extend to 32 bit by using a zero extend.

To add an instruction to the instruction set you need to add the following.

- Add elements to the data path: For example: to perform **ori**, zero extend unit is used to extend the immediate value from 16 bit to 32 bit.
- Add control elements: This new unit requires a new control signal to select the zero extend function (0) or the sign extend function (1). We will label the new signal ExtOp. Also, the 2-bit control signal ALUOp only encodes the operations add and subtract. Adding a third bit would allow the encoding of the operations AND and OR.

ALUOp [2]	ALUOp [1]	ALUOp [0]	Instruction
0	0	0	Memory operation (load, store)
0	0	1	Beq (subtract)
0	1	0	Ori
1	x	x	r-type

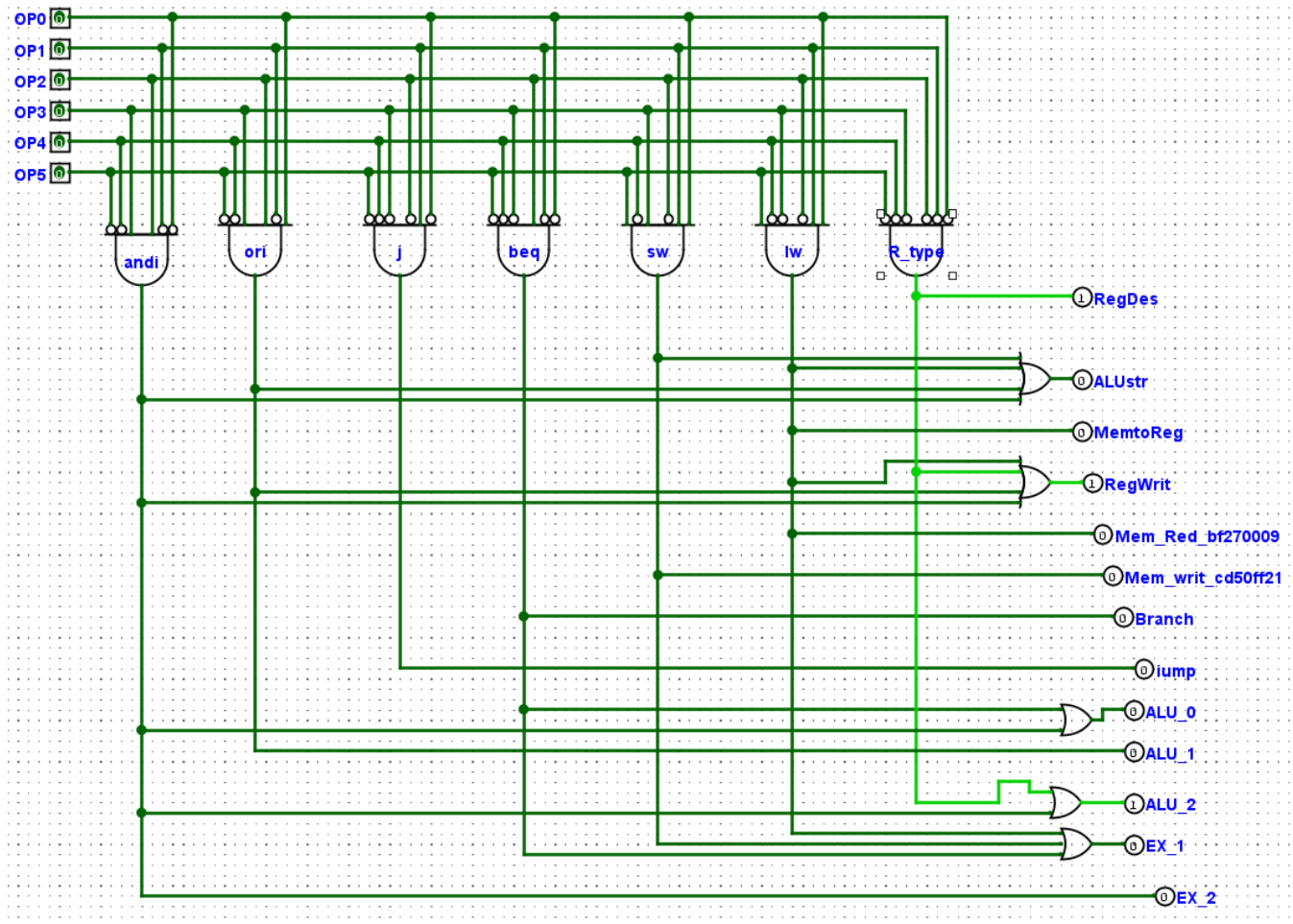
So, the ALU control table will change as follows:

Instruction	ALUOp	Funct	Alu operation	Alu control Input
lw	000	xxxxxx	Add	010
sw	000	xxxxxx	Add	010
beq	001	xxxxxx	Subtract	110
ori	010	xxxxxx	OR	001
add	100	100000	Add	010
sub	100	100010	Subtract	110
and	100	100100	And	000
or	100	100101	Or	001
slt	100	101010	slt	111

And the table of signals will change as follows:

Instr.	RegDst	ALUSrc	MemToReg	RegWrite	MemRead	MemWrite	Branch	AluOp	jump	ExtOp
R	1	0	0	1	0	0	0	100	0	x
lw	0	1	1	1	1	0	0	000	0	1
sw	x	1	x	0	0	1	0	000	0	1
beq	x	0	x	0	0	0	1	001	0	1
j	x	x	x	0	0	0	0	xxx	1	x
ori	0	1	0	1	0	0	0	010	0	0

The Design of Control Unit with ori and addi is as follow



Lab Task

- Use logisim to Implement control unit design.
- As a Home work, From previous labs, extend your 4 bit ALU and register file to 32 bit and connect it together.