KONYA FOOD AND AGRICULTURE UNIVERSITY FACULTY OF ENGINEERING AND ARCHITECTURE

COMPUTER ENGINEERING DEPARTMENT

CENG-3010 Computer Organization
Project 2.1

SOAT ARCHITECTURE DESIGN PROPOSAL

by

Sabriye Nur Şentürk-202010020011

Damla Uslu-202010020018

Enes Furkan Özdemir-202010020020

"in this project, you are going to design your own 8-bitMIPS like processor with the following features.

The processor should have at least 8 general purpose registers having 8 bits each.

The instruction size should be 16 bits.

You should include PC, hi, lo registers as in MIPS.

The size of the data memory and instruction memory should be 256 bytes each.

-conditional and unconditional branches should be supported.

-The processor should support procedure calls and implementations

- The processor should include at least the following instructions:add,sub, and, or, addi, andi, ori, xor,slt, slti,j,jr, jal, beq, bne,mul, muli ,lw,sw,mfhı,mflo"

SOAT INSTRUCTION SET

	NAME	MNEMONIC	FORMAT	OPERATION	OPCODE	FUNCTION CODE
1	Add	add	R	R[rd] = R[rs] + R[rt]	0000	000
2	And	and	R	R[rd] = R[rs] & R[rt]	0000	001
3	Subtract	sub	R	R[rd] = R[rs] - R[rt]	0000	010
4	Or	or	R	R[rd] = R[rs] | R[rt]	0000	011
5	Xor	xor	R	R[rd] = R[rs] ^ R[rt]	0000	100
6	Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])	0000	101
7	Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	0000	110
8	Jump Register	jr	R	PC=R[rs]	0000	111
9	Multiplication	mul	R	{Hi,Lo} = R[rs] * R[rt]	0001	000
10	Move From Hi	mfhi	R	R[rd] = Hi	0001	001
11	Move From Lo	mflo	R	R[rd] = Lo	0001	010
12	Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	0010	-
13	And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	0011	-
14	Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	0100	-
15	Multiplicate Immediate	muli	I	{Hi,Lo} = R[rs] * SignExtImm	0101	-
16	Branch on Equal	beq	I	if(R[rs]==R[rt]) PC=PC+2+BranchAddr	0110	-
17	Branch on not equal	bne	I	if(R[rs]!=R[rt]) PC=PC+2+BranchAddr	0111	-
18	Set Less Than Immediate	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	1000	-
19	Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	1001	-
20	Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	1010	-
21	Jump	j	J	PC=JumpAddr	1011	-
22	Jump and Link	jal	J	R[7]=PC+2;PC=JumpAddr	1100	-

23	**Shift Left Logical**	**sll**	**D**	**R[rd] = R[rt] << shamt**	**1101**	**-**
24	**Shift Right Logical**	**srl**	**D**	**R[rd] = R[rt] >> shamt**	**1110**	**-**
25	**Shift Right Arithmetic**	**sra**	**D**	**R[rd] = R[rt] >>> shamt**	**1111**	**-**

INSTRUCTION FORMATS

Instruction means a direction or an order that is given to the computer to execute. In our project, we used 25 instructions within four distinct instruction types.

- The R-type format is used for instructions with three register operands and an operation specified by a function code.
- The I-type format is used for instructions with two register operands and an immediate value.
- The J-type format is used for jump instructions with a large address space.
- The D-type format is used for shift instructions where one of the operands is a shift amount.

- **R-Type Instructions:**

  Instruction Format that we used for R-type is 16-bit sized as follows:

4-bits	3-bits	3-bits	3-bits	3-bits
OPCode	**rs**	**rt**	**rd**	**Funct**

Example:

add $t1, $t2, $t3: This instruction would add the contents of $t2 and $t3 and store the result in $t1.

- **I-Type Instructions:**

  Instruction Format that we used for I-type is 16-bit sized as follows

4-bits	3-bits	3-bits	6-bits
OPCode	**rs**	**rt**	**Immediate**

Example:

addi $t1, $t2, 10: This instruction would add the immediate value 10 to the contents of $t2 and store the result in $t1.

- **J-Type Instructions:**

  Instruction Format that we used for J-type is 16-bit sized as follow:

4-bits	12-bits
OPCode	Address

Example: j 1000: This instruction would jump to address 1000.

- **D-Type Instructions:**

  Instruction Format that we used for shift amount, D-type is 16-bit sized as follows:

4-bits	3-bits	3-bits	3-bits	3-bits
OPCode	rs	rt	rd	shamt

Example:

sll $t1, $t2, 2: This instruction would shift the contents of $t2 left by 2 bits and store the result in $t1.

Opcode: The operation code (opcode) is a 4-bit field that specifies the operation to be performed by the instruction. It determines the type of operation (arithmetic, logical, branch, etc.) and the format of the instruction (R, I, J, or D type).

rs: The source register (rs) is a 3-bit field that specifies the first source register operand. This register provides one of the values used in the operation.

rt: The second source register (rt) is a 3-bit field that specifies the second source register operand. This register provides another value used in the operation. In I-type instructions, rt is the destination register for the result.

rd: The destination register (rd) is a 3-bit field that specifies the register where the result of the operation will be stored. This field is used in R-type instructions.

Address: The address field is a 12-bit field used in J-type (jump) instructions. It specifies the address to jump to, allowing for a large range of jump addresses.

Immediate: The immediate field is a 9-bit field used in I-type instructions. It contains a constant value (immediate value) that is used in arithmetic or logical operations or as an offset for memory access.

funct: The function code (funct) is a 3-bit field used in R-type instructions. It specifies the exact operation to be performed within the category defined by the opcode.

shamt: The shift amount (shamt) is a 3-bit field used in D-type instructions for shift operations. It specifies the number of bit positions to shift.

REGISTERS

Registers are a list of names that program counter and variable names and numbers recorded.

8 general-purpose registers: $zero, $t1, $t2, $t3, $t4, $t5, $t6, $t7,$ra (return address)

Special registers: Hi (high result of multiplication), Lo (low result of multiplication), PC (program counter)

Index Num	Register
GPR	
0	$zero
1	$t1
2	$t2
3	$t3
4	$t4
5	$t5
6	$t6
7	$ra
SPR	
8	Hi
9	Lo
-	PC

CONTROL UNIT & DATAPATH:

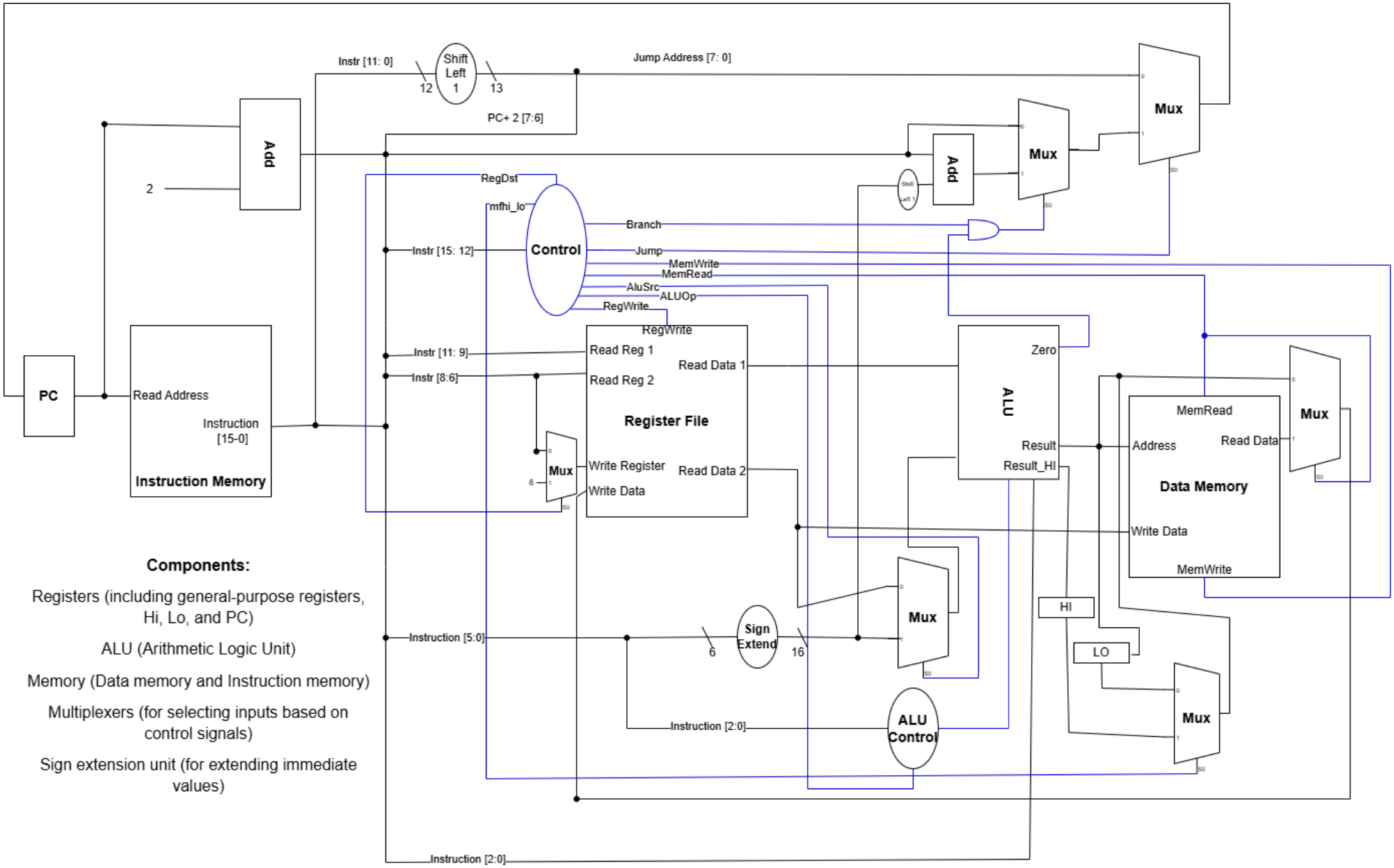
The control unit can be divided into two main parts:

1. Instruction Decoder: Decodes the 16-bit instruction to identify the operation, source/destination registers, and immediate values.
2. Control Signal Generator: Generates the necessary control signals for each part of the data path based on the decoded instruction.

Control Signals

- **RegDst:** Selects the destination register field (0 for rt, 1 for rd).
- **RegWrite:** Enables writing to the register file.
- **ALUSrc:** Selects the second ALU operand (register or immediate).
- **ALUOp:** Specifies the ALU operation.
- **MemRead:** Enables reading from data memory.
- **MemWrite:** Enables writing to data memory.
- **MemToReg:** Selects the value to write to the register (ALU result or memory data).
- **Jump:** Indicates a jump instruction.
- **Branch:** Indicates a branch instruction.
- **HiWrite, LoWrite:** Enable writing to Hi and Lo registers for multiplication.
- **MFHI_LO:** Controls read from HI or LO register (0 for HI, 1 for LO).

Instruction	RegDst	Jump	Branch	MemRead	MemWrite	MemtoReg	ALUOp	ALUSrc	RegWrite	MfhiLo_
add	1	0	0	0	0	0	0010	0	1	X
sub	1	0	0	0	0	0	0110	0	1	X
and	1	0	0	0	0	0	0000	0	1	X
or	1	0	0	0	0	0	0001	0	1	X
xor	1	0	0	0	0	0	1001	0	1	X
nor	1	0	0	0	0	0	1010	0	1	X
slt	1	0	0	0	0	0	0111	0	1	X
lw	0	0	0	1	0	1	0010	1	1	X
sw	X	0	0	0	1	X	0010	1	0	X
beq	X	0	1	0	0	X	0110	0	0	X
bne	X	0	1	0	0	X	0110	0	0	X
jump	X	1	0	0	0	X	X	X	0	X
jal	X	1	0	0	0	X	X	X	0	X
jr	X	0	0	0	0	X	X	X	0	X
mul	1	0	0	0	0	0	1000	0	1	X
mfhi	X	0	0	0	0	0	X	X	1	1
mflo	X	0	0	0	0	0	X	X	1	0
sll	1	0	0	0	0	0	1100	0	1	X
srl	1	0	0	0	0	0	1101	0	1	X
sra	1	0	0	0	0	0	1100	0	1	X
addi	0	0	0	0	0	0	0010	1	1	X
andi	0	0	0	0	0	0	0000	1	1	X
ori	0	0	0	0	0	0	0001	1	1	X
muli	0	0	0	0	0	0	1000	1	1	X
slti	0	0	0	0	0	0	0111	1	1	X



Components:

Registers (including general-purpose registers, Hi, Lo, and PC)

ALU (Arithmetic Logic Unit)

Memory (Data memory and Instruction memory)

Multiplexers (for selecting inputs based on control signals)

Sign extension unit (for extending immediate values)