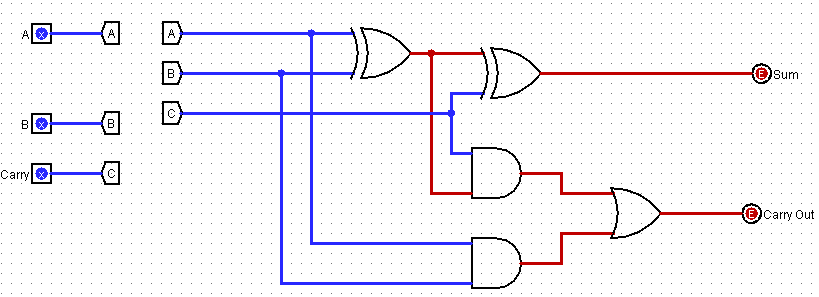
**Damla Övek - 31587**

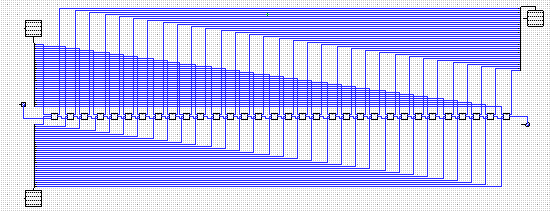
**Dilara Yıldırım - 32446**

**COMP303 TERM PROJECT:**

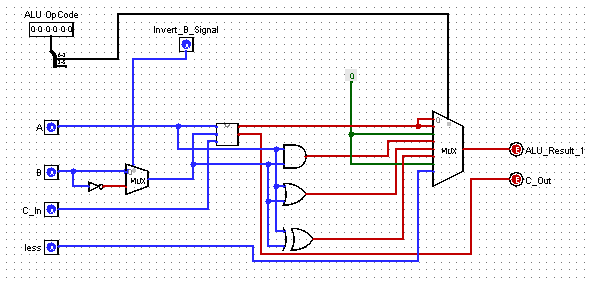
1-bit adder given by the project:



32-bit (ripple carry) adder constructed by using 1-bit adders:

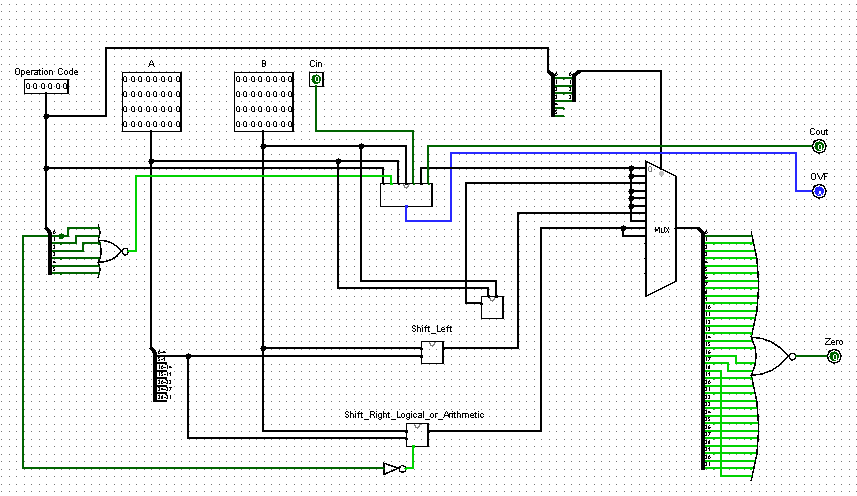


1-bit ALU:



32-bit ALU:

In the following picture, we can see the ALU with 32 bits including Multiplier and three different shifters.



|  |  |  |
| --- | --- | --- |
| OPCODE | Instruction | Operation |
| 000000 | sub | C = A - B |
| 000001 | add | C = A + B |
| 000010 | mult | C = A \* B |
| 000011 | and | C = A & B |
| 000100 | or | C = A | B |
| 000101 | xor | C = A ^ B |
| 000110 | sll | (shift left logical) C = A << B |
| 000111 | slt | (set less than) C = 0 IF (A >= B) ; 1 IF (A < B) |
| 001000 | sra | (shift right arithmetic) C = A >>> B |
| 001001 | srl (shift right logical) | C = A >> B |

**MAIN CONTROL UNIT:**

In this unit, we must generate all control signals such as RegDst, ALUSrc, MemtoReg and etc. In the generation of these signals operation code of each instruction class is used. For example, if the instruction is an R-Type instruction, the operation code will be 000000. However, the ALU operation code inputs will be given as in the project description. For example, for an R-Type instruction, an ADD instruction, the opcode will be **000001** only as an input to the ALU. This means, ALU will make addition, if the ALU operation input is **000001.**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Inst | O5 | O4 | O3 | O2 | 01 | O0 | RegDst | ALUSrc | MemtoReg | RegWrt | MemR | MemW | beq | bne | bge | J | ALUOp1 | ALUOp0 |
| R-ty | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | X |
| lw | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| sw | 0 | 0 | 1 | 0 | 1 | 1 | X | 1 | X | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| addi | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| beq | 0 | 0 | 1 | 1 | 0 | 0 | X | 0 | X | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| bne | 0 | 0 | 1 | 1 | 0 | 1 | X | 0 | X | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| bge | 0 | 0 | 1 | 1 | 1 | 1 | X | 0 | X | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| j | 0 | 0 | 1 | 1 | 1 | 0 | X | X | X | 0 | 0 | 0 | 0 | 0 | 0 | 1 | X | X |

# lw $3, 0($2)

14: 8c430000; 1000 1100 0100 0011 0000 0000 0000 0000

100011 00010 00011 0000000000000000

# lw $3, 0($5) for our ınstruction type

001010 00101 00011 0000000000000000

0010 1000 1010 0011 0000000000000000

28A30000

RegDst = O5’O4’O3’O2’O1’O0’

ALUSrc = O5’O4’O3O2’O1 + O5’O4’O3O2’O1’O0’

MemtoReg = O5’O4’O3O2’O1O0’

RegWrt = O5’O4’O3’O2’O1’O0’ + O5’O4’O3O2’O1O0’ + O5’O4’O3O2’O1’O0’

MemRead = O5’O4’O3O2’O1O0’

MemWrite= O5’O4’O3O2’O1O0

Beq= O5’O4’O3O2O1’O0’

Bne= O5’O4’O3O2O1’O0

Bge= O5’O4’O3O2O1O0

Jump= O5’O4’O3O2O1O0’

ALUOp1= O5’O4’O3’O2’O1’O0’

ALUOp0= O5’O4’O3O2O1’O0’ + O5’O4’O3O2O1’O0 + O5’O4’O3O2O1O0

**ALU CONTROL UNIT:**

In this unit, the operation signals for ALU unit is generated. According to these signals ALU selects the correct operation, whether it is a ADD, SUB, OR etc. There are two different inputs for this unit. The first one is the ALU-OP (2-bits) which come from the Main Unit. With these two bits, the class of instruction type is given to the ALU. Another input field is ‘FunctField’ which is a directly a part of the instruction. This field is only used for R-Type instructions; for other type of instructions, this field can be ignored.

As a new instruction, we added ‘bge’ which stands for “BRANCH GREATER THAN OR EQUAL TO” instruction. This is a new branch instruction which can be used as “IF A>=B THEN GOTO LABEL”.

To design this instruction, we compare both operands by subtracting one from another, here we check if the ALU result is greater than or equal to zero. If it is then, branch condition is OK. This means, if the instruction is branch (i.e. bge) and the ALU result is greater than or equal to zero, then branch is done. To achieve this, we control both “ALU Zero” output and “ALU result” itself. If the result is non-negative, which means MSB bit of result is 0 or “ALU Zero” output 1, then branch condition is OK.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operation Code | ALUOp  (A) | Instruction Explanation | FunctField  (F) | ALU Function | ALU Control Opcode (6-bits) |
| R-type | 10 | sub | 100010 | sub | 000000 |
| R-type | 10 | add | 100000 | add | 000001 |
| R-type | 10 | mult | 011000 | mult | 000010 |
| R-type | 10 | and | 100100 | and | 000011 |
| R-type | 10 | or | 100101 | or | 000100 |
| R-type | 10 | xor | 100110 | xor | 000101 |
| R-type | 10 | sll (shift left logical) | 000000 | sll | 000110 |
| R-type | 10 | slt (set less than) | 101010 | slt | 000111 |
| R-type | 10 | sra (shift right arithmetic) | 000011 | sra | 001000 |
| R-type | 10 | srl (shift right logical) | 000010 | srl | 001001 |
| lw | 00 | Load word | XXXXXX | add | 000001 |
| sw | 00 | Store word | XXXXXX | add | 000001 |
| addi | 00 | Add immediate value | XXXXXX | add | 000001 |
| beq | 01 | Branch eq. | XXXXXX | sub | 000000 |
| bne | 01 | Branch not eq. | XXXXXX | sub | 000000 |
| bge | 01 | Branch greater than | XXXXXX | sub | 000000 |

ALU\_Control\_Out(0) = A1A0’F5F4’F3’F2’F1’F0’+ A1A0’F5F4’F3’F2F1’F0’+ A1A0’F5F4’F3’F2F1F0’+ A1A0’F5F4’F3F2’F1F0’+ A1A0’F5’F4’F3’F2’F1F0’+A1’A0’

ALU\_Control\_Out(1) = A1A0’F5’F4F3F2F1F0 + A1A0’F5F4’F3’F2F1’F0’ + A1A0’F5’F4’F3’F2’F1’F0’+ A1A0’F5F4’F3F2’F1F0’

ALU\_Control\_Out(2) = A1A0’F5F4’F3’F2F1’F0 + A1A0’F5F4’F3’F2F1F0’+ A1A0’F5’F4’F3’F2’F1’F0’+ A1A0’F5F4’F3F2’F1F0’

ALU\_Control\_Out(3) = A1A0’F5’F4’F3’F2’F1

ALU\_Control\_Out(4) = 0

ALU\_Control\_Out(5) = 0