A SIMPLE 24-bit COMPUTER SYSTEM

By

Nikko Brian Caraan

Carlos Rafael Catalan

Melvin Flores

Klarence Manangan

CMSC 132 T-3L

Computer Architecture

Our group is asked to create a simple computer that can support a 24-bit input with the first 8-bits dedicated to an instruction and the remaining 16 bits for two operands with 8 bits each. We will have 8 general purpose registers (named r1,r2,...,r8), 2 8-bit Memory Address Register, 1 8-bit Program Counter, 1 8-bit Instruction Counter and 1 8-bit Flag functioning as zero flag and sign flag(the first most significant bits for the zero flag and the last 4 bits for the sign flag). The group is also asked to provide a mapping of the opcode to the machine code, assign a register to a register number in binary and specify the chosen ALU, Data path and Control Unit Design.

Mapping of opcode to Machine Code

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Constants** | **Assembly Instruction** | **First Byte** | **Second Byte** | **Third Byte** | **Operation** |
| **LOAD** | **LOAD Rn, direct** | **00000000** | Rn | Direct | Rn ⭠ M(direct) |
| **STORE** | **STORE direct, Rn** | **00000001** | Direct | Rn | M(direct) ⭠ Rn |
| **SAVE** | **SAVE Rn, Immediate (?)** | **00000010** | Rn | Immediate | Rn ⭠ |
| **INC** | **INC Rn, Rm** | **00000011** | Rn | Rm | Rn ⭠ Rn + Rm (1) |
| **DEC** | **DEC Rn, Rm** | **00000100** | Rn | Rm | Rn ⭠ Rn – Rm (1) |
| **ADD** | **ADD Rn, Rm** | **00000101** | Rn | Rm | Rn ⭠ Rn + Rm |
| **SUB** | **SUB Rn, Rm** | **00000110** | Rn | Rm | Rn ⭠ Rn – Rm |
| **MUL** | **MUL Rn, Rm** | **00000111** | Rn | Rm | Rn ⭠Rn \* Rm |
| **DIV** | **DIV Rn, Rm** | **00001000** | Rn | Rm | Rn ⭠Rn/Rm |
| **CMP** | **CMP Rn, Rm** | **00001001** | Rn | Rm | Rn ⭠Rn-Rm |
| **AND** | **AND Rn, Rm** | **00001010** | Rn | Rm | Rn ⭠Rn and Rm |
| **OR** | **OR Rn, Rm** | **00001011** | Rn | Rm | Rn ⭠Rn or Rm |
| **NOT** | **NOT Rn, Rm** | **00001100** | Rn | Rm | Rm ⭠ not Rn |
| **XOR** | **XOR Rn, Rm** | **00001101** | Rn | Rm | Rn ⭠ Rn xor Rm |
| **JE** | **JE Rn, Rm** | **00001110** |  |  |  |
| **JG** | **JG Rn, Rm** | **00001111** |  |  |  |
| **JL** | **JL Rn, Rm** | **00010000** |  |  |  |
| **JMP** | **JMP Rn, Rm** | **00010001** |  |  |  |

THE INSTRUCTON SET

Data Transfer Instructions

**LOAD**

Format: **LOAD** <register>, <memory address>

Description: Loads data from the specified memory address to the given register

**STORE**

Format: **STORE** <memory address>, <register>

Description: Stores data from the specified register address to the given memory address

**SAVE**

Format: **SAVE** <register>, <memory address>

Description: Loads data from the specified memory address to the given register

Arithmetic Instructions

**INC**

Format: **INC** <register1>, <register2>

Description: Increments the value in register2 and saves the result to register1

**DEC**

Format: **DEC** <register1>, <register2>

Description: Decrements the value in register2 and saves the result to register1

**ADD**

Format: **ADD** <register1>, <register2>

Description: Adds the value of register1 to the register2 and stores the result in register1

**SUB**

Format: **SUB** <register1>, <register2>

Description: Subtracts the value of register2 to the register1 and stores the result in register1

**MUL**

Format: **MUL** <register1>, <register2>

Description: Multiply the value of register1 by register2 and stores the result in register1

**DIV**

Format: **DIV** <register1>, <register2>

Description: Divides the value of register1 by register2 and stores the result in register1

Comparison Instructions

**CMP**

Format: **CMP** <register1>, <register2>

Description: Multiply the value of register1 by register2 and stores the result in register1

Logic Instructions

**AND**

Format: **AND** <register1>, <register2>

Description: Performs bitwise AND operation between register1 and register2 and stores the result in register1

**OR**

Format: **OR** <register1>, <register2>

Description: Performs bitwise OR operation between register1 and register2 and stores the result in register1

**NOT**

Format: **NOT** <register1>, <register2>

Description: Performs bitwise NOT operation on register1 and stores the result in register2

**XOR**

Format: **XOR** <register1>, <register2>

Description: Performs bitwise XOR operation between register1 and register2 and stores the result in register1

Jump Instructions

**JE**

Format: **JE** <register1>, <register2>

Description:

**JG**

Format: **JG** <register1>, <register2>

Description:

**JL**

Format: **JL** <register1>, <register2>

Description:

**JMP**

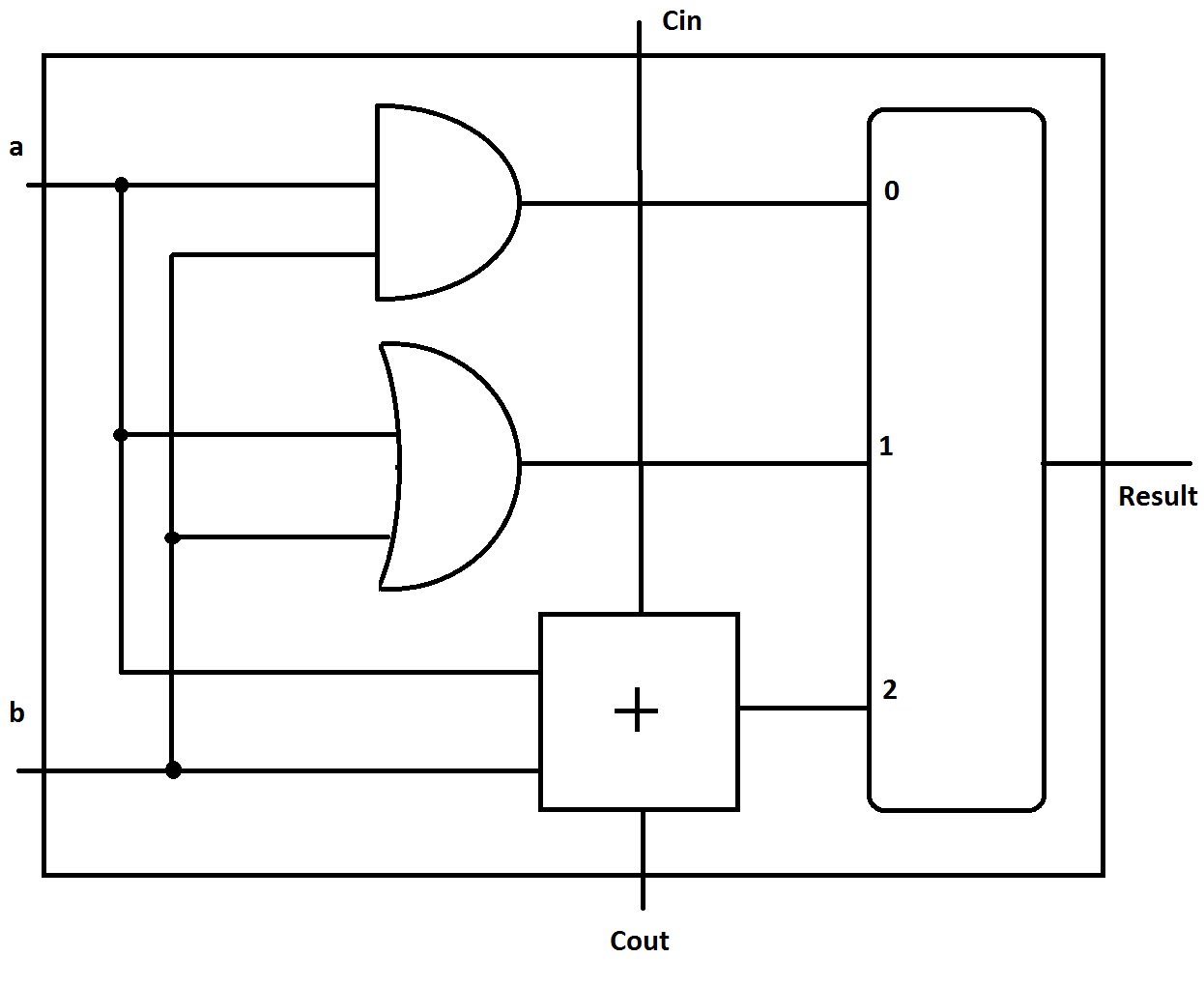
Format: **JMP** <register1>, <register2>

Description:

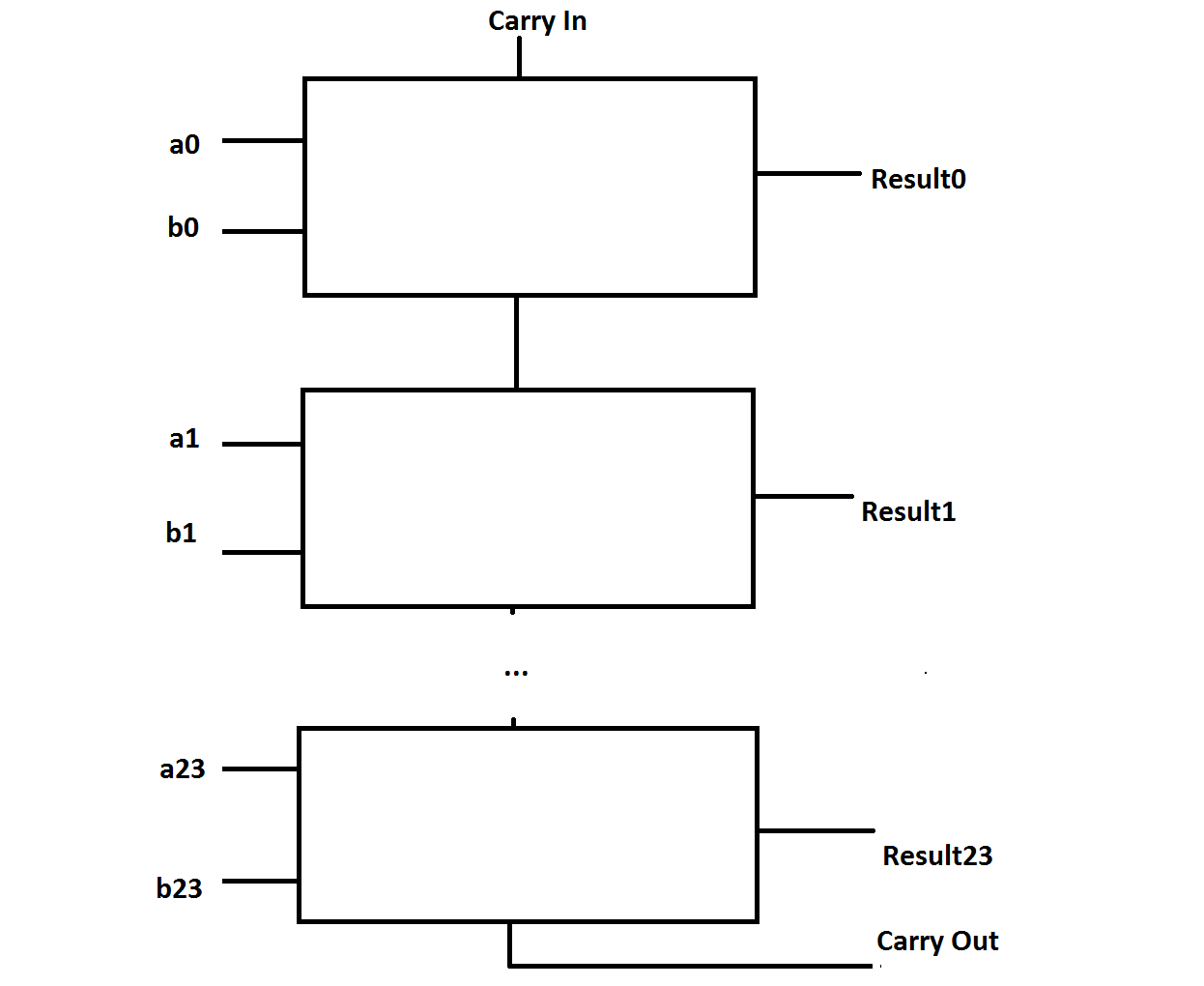
|  |  |
| --- | --- |
| Constants | Assembly Instruction |
| LOAD | LOAD Rn, direct |
| STORE | STORE direct, Rn |
| SAVE | SAVE Rn, Immediate (?) |
| INC | INC Rn, Rm |
| DEC | DEC Rn, Rm |
| ADD | ADD Rn, Rm |
| SUB | SUB Rn, Rm |
| MUL | MUL Rn, Rm |
| DIV | DIV Rn, Rm |
| CMP | CMP Rn, Rm |
| AND | AND Rn, Rm |
| OR | OR Rn, Rm |
| NOT | NOT Rn, Rm |
| XOR | XOR Rn, Rm |
| JE | JE Rn, Rm |
| JG | JG Rn, Rm |
| JL | JL Rn, Rm |
| JMP | JMP Rn, Rm |

THE ALU

Since we are designing a 24-bit computer, the ALU therefore would accept 24-bit input.

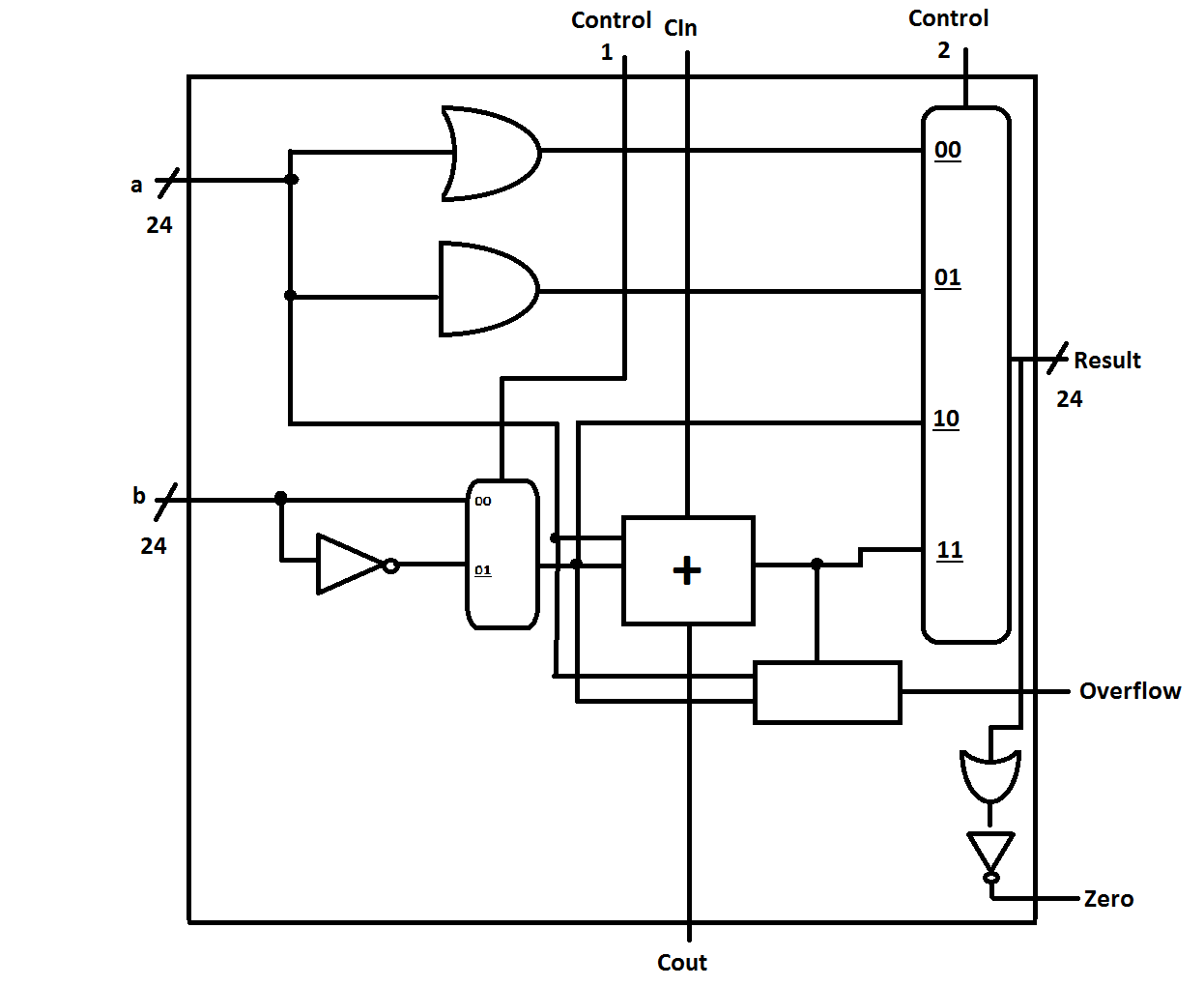


1-bit ALU



24-bit ALU

A number of functions are performed internally but only one result is chosen as the output of the ALU. A 24- bit ALU is just the compilation of 24 identical 1-bit ALUs.



Data Path Design

R1s

R2s

RFs

To Control Unit

MEMORY

ALU

RF()

MUX

MUX

MUX

Control Unit Design

Conclusion

**References**

ALU Design. (n.d.). Retrieved May 2, 2016, from http://www.utdallas.edu/~poras/courses/ee3320/xilinx/upenn/lab5-ALUDesign.htm

Manuales P., Payoyo, N. Garcia, G. & Santos, J. (n.d.) A Simple Computer System: Implementation in VHDL.