A picture containing circuit, electronics

Description automatically generated

**4+2-bit Opcode Computer**

**Logo

Description automatically generated**

**Khizar Ali Shah - 2020196**

**Syed Muhammad Ashhar - - 2020478**

**Muhammad Omer - 2020335**

**Harris Abdullah - 2020159**

**Table of Contents**

[Abstract: 3](#_Toc104359473)

[Instruction Format: 4](#_Toc104359474)

[Addressing Modes: 4](#_Toc104359475)

[Direct Address: 4](#_Toc104359476)

[Indirect Address: 4](#_Toc104359477)

[Indexed Relative Address: 4](#_Toc104359478)

[Immediate Address: 5](#_Toc104359479)

[Registers: 5](#_Toc104359480)

[Main Memory: 7](#_Toc104359492)

[Flags: 7](#_Toc104359493)

[Extended Accumulator (E): 7](#_Toc104359494)

[Interrupt Flip-flop (R): 7](#_Toc104359495)

[Start Flip-flop (S): 7](#_Toc104359496)

[Common Bus: 8](#_Toc104359497)

[Instruction Cycle 9](#_Toc104359498)

[Fetch: 9](#_Toc104359499)

[Decode: 9](#_Toc104359500)

[Execute: 9](#_Toc104359501)

[Memory Reference Instructions: 10](#_Toc104359502)

[Execution of Memory Reference Instructions: 10](#_Toc104359503)

[Register Reference Instructions: 11](#_Toc104359504)

[I/O Instructions: 11](#_Toc104359505)

[Instruction Cycle Flowchart: 12](#_Toc104359506)

[Arithmetic Unit: 13](#_Toc104359507)

[Logic Unit: 14](#_Toc104359508)

[Arithmetic-Logic-Shift Unit: 14](#_Toc104359509)

[Multiply Circuit: 15](#_Toc104359510)

[Example: 16](#_Toc104359511)

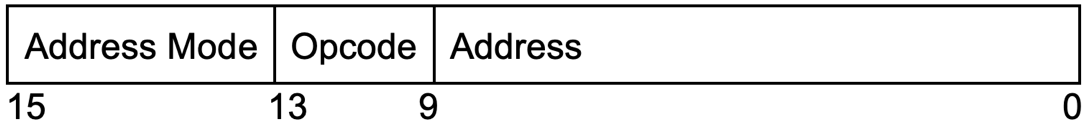
[References: 17](#_Toc104359512)

# Abstract:

In this project report, a basic computer is designed which can perform basic computation like arithmetic, logic, and shift. It works on the stored-program concept coined by von Newmann. The instruction format is 16-bits which is divided into three parts. These include the 2-bits of mode, 4-bits for opcode, and 10-bits for the address. The 4-bit opcode defines a total of 16 instructions, 15 of which are categorized into memory reference instructions. Remaining opcode opens the whole category of register and input/output instructions. The novelty in this design is the multiplication circuit. This circuit can perform multiplication in a single clock pulse, which eliminates the need of loading and storing many operands in many clock cycles. This is a fast multiplication method inspired from a mathematics book given in the reference. Although there can be some propagation delay of which need a wait of some clock cycles, still if the efficient hardware is used the circuit can work much better. The Arithmetic unit contains basic arithmetic instructions. Logic unit contains fundamental logic instructions while the shift unit contains the right and left shift instructions. There three units are combined to form an ALSU which communicates with the system common bus, accumulator, and data register for data-flow.

# Instruction Format:

The instruction is 16-bits long/wide, and it consists of 2-bits for mode, 4 for opcode and remaining 10 for address/operand. There are four addressing modes: direct address, indirect address, register address, and immediate address. The code for direct addressing is 00, indirect addressing is 01, relative addressing is 10, and immediate is 11.



# Addressing Modes:

## Direct Address:

Direct addressing means that the operand is in the memory space specified by the address stored in the lower 10 bits of the 16-bit instruction. It is direct address when mode bits are 00.

## Indirect Address:

Indirect addressing means that operand is present in the address specified by the data-field of the address present in the lower 10 bits of the 16-bit instruction. A 01 in the mode field of the instruction indicates an indirect address.

## Indexed Relative Address:

In this mode of addressing, the address part of the instruction does not contain the correct operand address; and rather it has to be added to the address stored in Index Register, i.e., the contents of the Index Register, to actually specify the address of the operand. Where does the contents of Index Register come from? They are specified by the programmer. Hence there is a specific instruction that gives the programmer a flexibility to change the value of this register, giving the computer access to those parts of the memory which is not accessible through a simple 10-bit address, like in the case of having multiple memory chips. The code for indexed relative address mode is 10.

## Immediate Address:

Immediate addressing means operand is present in these 10 bits (lower bits of the instruction) itself. So there is no need to reach the memory for accessing the operand.

# Registers:

The following registers are present in our computer:

|  |  |  |
| --- | --- | --- |
| Program Counter | PC | 10-bits |
| Instruction Register | IR | 16-bits |
| Data Register | DR | 16-bits |
| Temporary Register | TR | 10-bits |
| Accumulator | AC | 16-bits |
| Address Register | AR | 10-bits |
| Input Register | INP | 8-bits |
| Output Register | OUT | 8-bits |
| Mode Register | MR | 2-bits |
| Multiply Result Register | MM | 32-bits |
| Index Register | XR | 16-bits |

## Program Counter (PC):

It stores the address of the next instruction to be fetched from the main memory. Since its contents are a memory address, it is 10-bits in size.

## Address Register (AR):

It stores the address of the operand which is present in memory. Since the address is a memory address, its size is 10-bits.

## Instruction Register (IR):

The contents of IR are the current instruction to be decoded and executed by the processor. It contains a complete instruction; hence its size is 16-bits.

## Data Register (DR):

DR is used to store the operand. This operand is used for computation, either alone or with the second operand present in AC. Operand’s size is 16-bits and so is DR.

## 

## Temporary Register (TR):

It holds the address of PC temporarily when an interrupt or a subroutine occurs. This makes it possible to further stores this previous PC value, now stored in TR, to be saved in some address in memory specified by the new PC so that it can be retrieved later when the interrupt or subroutine has finished. It must have size 10-bits because of storing the address, not the complete instruction.

## Accumulator (AC):

It stores the operand which can be the result of a calculation. The operand is taken from memory, through DR, so it must be of size 16-bits.

## Input Register (INP):

When an input from the standard input device, here keyboard, comes then it is stored in this register. Each keyboard button has an ASCII value which is of 8-bits. So, for its storage, INP is of the same size. It will be further processed by the CPU.

## Output Register (OUT):

The value in OUT is the ASCII value of a character, hence 8-bits. The output device, say a monitor takes this value and shows it.

## Mode Register (MR):

It is used to specify the addressing modes. Since there are four addressing modes, it is of size 2-bits. The detail of these addressing modes is in section 3.

## Index Register (XR):

It is used for indexed addressing mode. It stores the address that is to be added to the address in AR to produce the effective address of operand. It can be changed by the programmer.

## Multiply Result Register (MM):

Its sole purpose is to store the result generated by the Multiply circuit. Its size is larger than all other registers because the result of multiplication of two n-bit numbers gives a number of size 2n or 2n-1 bits. In the present case, when two 16-bit numbers are multiplied, result can be of 32-bits. Hence its size is 32-bits.

# 

# Main Memory:

The memory size we chose has 210 words; each is 16-bit (2 bytes) long. Hence the memory size is 1024 x 2 B = 2048 bytes = 2 Kebi Byte. Since there are 210 words, the address line is 10 bits long, and since each word is 16-bit long, the data bus is 16 bits. It has one more line for ‘write’ and ‘read’ operations. A 1 on this line means WRITE while a 0 means READ.

The contents of memory, in a specified address, can be changed when write is on, i.e., 1. When write is on, the contents in data-bus will be placed in the address of memory whereas the address is given using address bus. In case read is on, the contents in memory in the specified address are placed on the data-bus so that they can be read by the processor.

Table

Description automatically generated

# Flags:

Flags are flip-flops that can contain 0 or 1 as their value. They are different from registers because each flag consists of a single flip-flop instead of multiple flip-flops. Flags are used to handle different cases during instruction cycle.

## Extended Accumulator (E):

When computations are done on the contents of AC, they can result in a number of size 1-bit larger than what AC can hold. This last bit is stored in the E flip-flop. It is usually called the overflow bit. Moreover, during circular shifts, it stores the LSB or MSB of AC so that it is not lost during the left or right shift and thus can be later taken and stored in proper position in AC to complete the circular shift.

## Interrupt Flip-flop (R):

It becomes 1 when an interrupt occurs. Otherwise, it is 0.

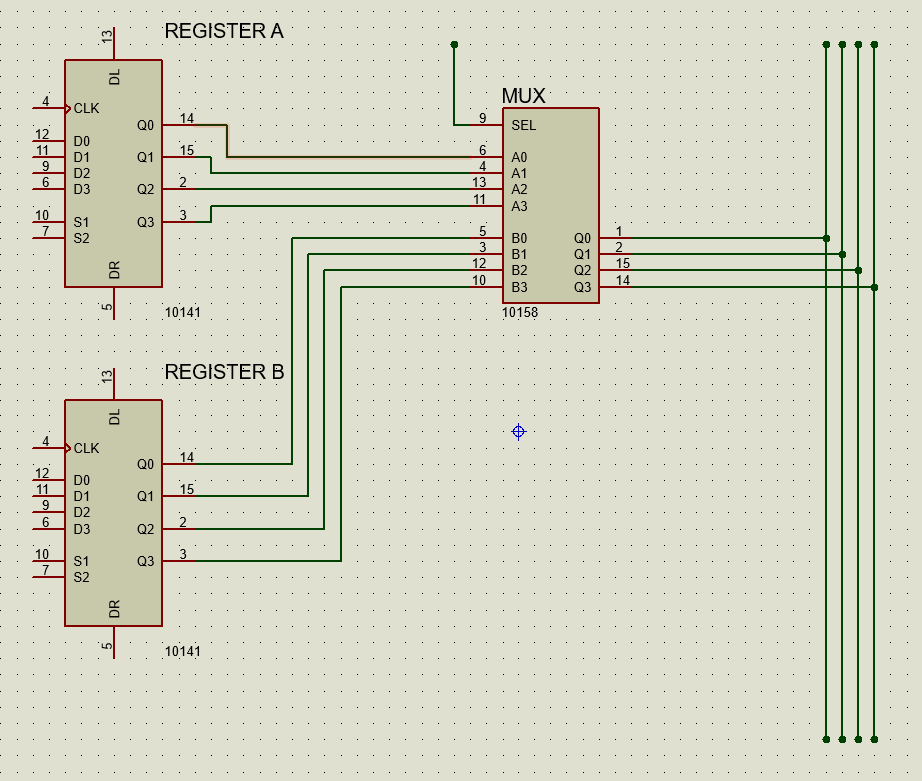
## Start Flip-flop (S):

It is 1 when the program is running. It becomes 0 when the program stops. It is different form the ring counter (RC) which loops back to T0 to start next instruction cycle when one instruction finishes. When S=1, instruction cycle comes to a halt.

# Common Bus:

Using a dedicated system bus for every pair of registers is very inefficient and not economical. Hence a common bus must be used. Since the maximum size of any register is 16-bits, the size of a common bus is 16-bits. The only register with larger size, 32-bits, is the MM, it is illogical to construct a 32-bit bus for this. The data from MM can be transferred using lower and upper 16-bits instead of all at once.

The bus is constructed as follows:



With similar design and logic, it can be extended from 4-bits to 16-bits. Since there are 11 registers, 4 select lines are needed to select these distinctly. Their configuration is as follows:

|  |  |  |
| --- | --- | --- |
| Register | Select Lines (S3 S2 S1 S0) | Hex Code |
| PC | 0000 | 0 |
| AR | 0001 | 1 |
| MM | 0010 | 2 |
| IR | 0011 | 3 |
| XR | 0100 | 4 |
| INP | 0101 | 5 |
| OUT | 0110 | 6 |
| DR | 0111 | 7 |
| TR | 1000 | 8 |
| AC | 1001 | 9 |
| MR | 1010 | 10 |

# Instruction Cycle

A digital computer runs programs that are present in the main memory. A program consists of data and a set of instructions placed in (generally) consecutive memory locations. These instructions run in a sequential manner, and each instruction is said to be executed completely when it has gone through a complete instruction cycle. An instruction cycle consists of fetch, decode, and execute. These are described in detail in the following:

## Fetch:

Program Counter, PC, stores the address of the next instruction that is to be fetched from memory. In the fetch phase, this address in PC is moved (parallel) to the register AR in a single clock pulse. Then in the next clock pulse, the PC must point to the next instruction, which is placed in the next memory location, so it is incremented by 1. Moreover, the memory M specified by the address in AR, i.e., M[AR], is accessed and the ‘read’ line is turned on. It placed the instruction (or data) in M[AR] to the system common bus. Then the contents of data-bus are loaded onto the Instruction Register IR. When an instruction or data is placed in IR, the fetch phase is completed.

AR←PC

IR←M[AR], PC←PC+1

## Decode:

The lower 10-bits are loaded onto AR. Next 4 bits, which is the opcode, is sent to the 4x16 decoder, part of the hardwired control unit, for generating control signal Ki. The two most significant bits in IR are moved to the Mode Register.

AR←IR[0:9], Decoder←IR[10:13], MR←IR[14:15]

In case of indirect address, the address specified by AR is visited and the value stored in memory, which is the address of operand, is brought to AR. In direct addressing, this process is not done because the address of operand is already present in AR. Similarly, AR contains the operand itself in case of immediate addressing. It must be moved to DR; in all other cases, the operand is stored in DR using this micro-operation at T4.

DR←M[AR]

## Execute:

The 4x16 decoder can generate any signal from K0 to K15 for the opcodes 0000 to 1111 respectively. Each Ki corresponds to an instruction. There are three types of instructions. The values from 0001 to 1111 represent memory reference instructions. When it becomes 0000, then it is used as either register or input/output instructions. When the LSB of Mode Register MR [0] is 0, then IO instructions are executed. When MR [0] is 1, then Register reference instructions are executed.

# Memory Reference Instructions:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Decoder  Signal | Instruction  Symbol | Hex Code | | | |
| Direct | Indirect | Immediate | Relative |
| MR=00 | MR=01 | MR=10 | MR=11 |
| K1 | XOR | 0 1 x x x | 1 1 x x x | 2 1 x x x | 3 1 x x x |
| K2 | OR | 0 2 x x x | 1 2 x x x | 2 2 x x x | 3 2 x x x |
| K3 | AND | 0 3 x x x | 1 3 x x x | 2 3 x x x | 3 3 x x x |
| K4 | JUN | 0 4 x x x | 1 4 x x x | NOT USED | 3 4 x x x |
| K5 | NOR | 0 5 x x x | 1 5 x x x | 2 5 x x x | 3 5 x x x |
| K6 | JSR (Jump Save Return Address) | 0 6 x x x | 1 6 x x x | NOT USED | 3 6 x x x |
| K7 | NAND | 0 7 x x x | 1 7 x x x | 2 7 x x x | 3 7 x x x |
| K8 |  | 0 8 x x x | 1 8 x x x | 2 8 x x x | 3 8 x x x |
| K9 | ADD | 0 9 x x x | 1 9 x x x | 2 9 x x x | 3 9 x x x |
| K10 | SUB | 0 A x x x | 1 A x x x | 2 A x x x | 3 A x x x |
| K11 | MLT | 0 B x x x | 1 B x x x | 2 B x x x | 3 B x x x |
| K12 | ISZ | 0 C x x x | 1 C x x x | NOT USED | 3 C x x x |
| K13 | STM (Store Multiply Result) | 0 D x x x | 1 D x x x | NOT USED | 3 D x x x |
| K14 |  | 0 E x x x | 1 E x x x | 2 E x x x | 3 E x x x |

# Execution of Memory Reference Instructions:

|  |  |  |
| --- | --- | --- |
| Symbolic Instruction | Control Signals | Micro-operations |
| XOR | K1.T5 | AC ← DR xor AC, RC←0 |
| OR | K2.T5 | AC ← DR ∨ AC, RC←0 |
| AND | K3.T5 | AC ← DR ∧ AC, RC←0 |
| JUN | K4.T5 | PC ← AR, RC←0 |
| NOR | K5.T5 | AC ← DR nor AC, RC←0 |
| JSR | K6.T5  K6.T6  K6.T7 | M[AR] ← PC  AR ←AR+1  PC ← AR, RC ← 0 |
| NAND | K7.T5 | AC ← (DR ∧ AC)’, RC←0 |
|  | K8.T5  K8.T6 |  |
| ADD | K9.T5 | AC ← DR + AC, RC←0 |
| SUB | K10.T5 | AC ← AC - DR, RC←0 |
| MLT | K11.T5 | MM ← DR \* AC, RC←0 |
| ISZ | K12.T5  (DR)’.K12.T6  K12.T6 | DR←DR+1  PC←PC+1  M[AR] ← DR, RC←0 |
| STM | K13.T5  K13.T6  K13.T7 | M[AR] ← MM[0:15]  AR ← AR+1  M[AR] ← MM[16:31], RC←0 |

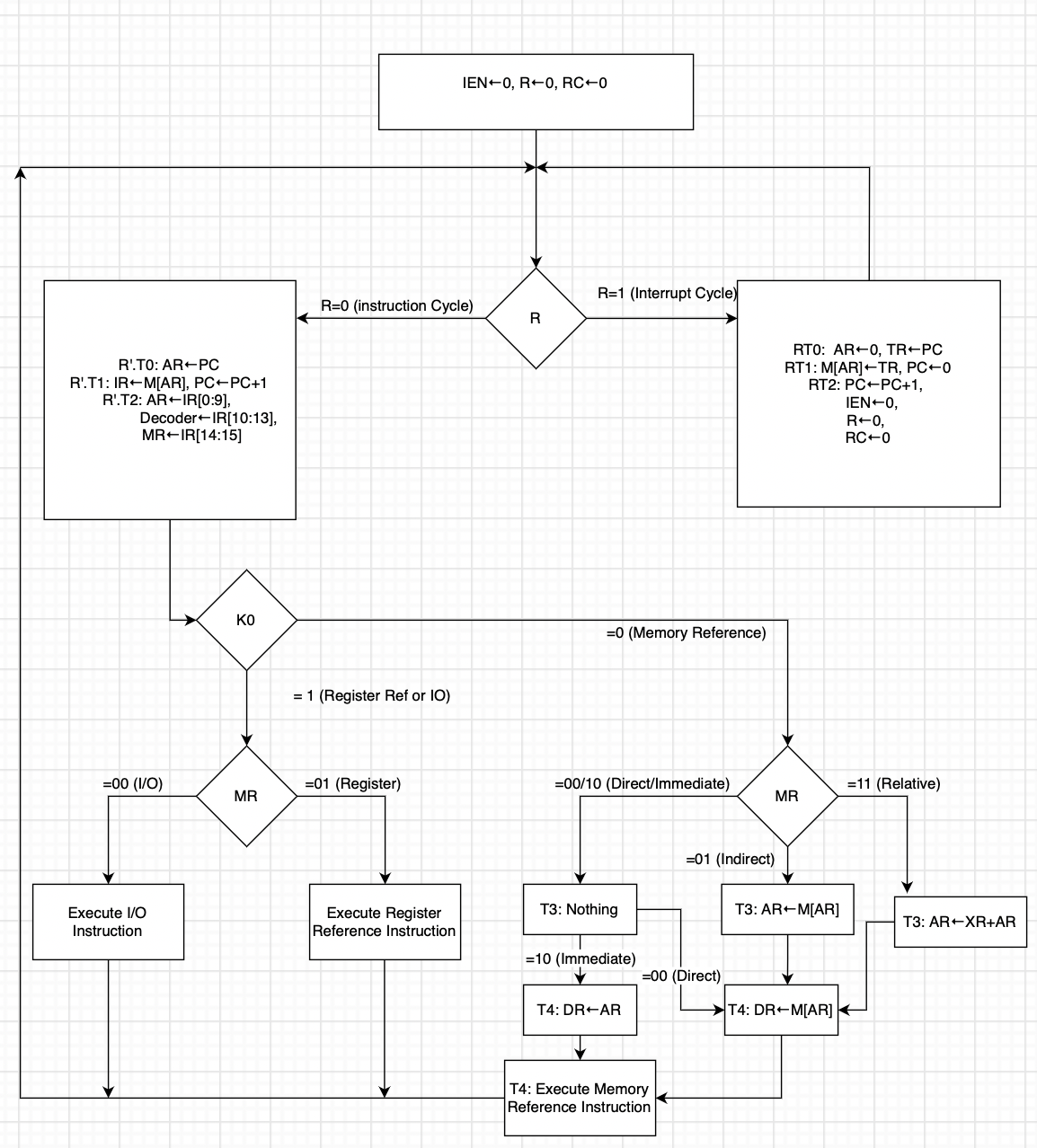
# Register Reference Instructions:

|  |  |  |  |
| --- | --- | --- | --- |
| Symbol | Meaning | Hex Code  MR[0] = 1 | Micro-operation  (RC←0 after every micro-operation) |
| CMA | Complement AC | 1 0 0 1 | AC←AC’ |
| CME | Complement E | 1 0 0 2 | E←E’ |
| INC | Increment AC | 1 0 0 3 | AC←AC+1 |
| CLA | Clear AC | 1 0 0 4 | AC←0 |
| CLE | Clear E | 1 0 0 5 | E←0 |
| CLP | Clear PC | 1 0 0 6 | PC←0 |
| DCA | Decrement AC | 1 0 0 7 | AC←AC-1 |
| DCP | Decrement PC | 1 0 0 8 | PC←PC-1 |
| INA | Increment AR | 1 0 0 9 | AR←AR+1 |
| CIR | Circular shift right | 1 0 0 A | E←AC[0], AC←shr(AC), AC[15] ←E |
| CIL | Circular shift left | 1 0 0 B | E←AC[15], AC←shl(AC), AC[0] ←E |
| SPA | Skip next instruction if AC [15] =0 | 1 0 0 C | If (AC[15]=0) then PC←PC+1 |
| SZA | Skip next instruction if AC=0 | 1 0 0 D | If (AC=0) then PC←PC+1 |
| SZE | Skip next instruction if E=0 | 1 0 0 E | If (E=0) then PC←PC+1 |
| HLT | Halt the Computer | 1 0 0 0 | S←0 |

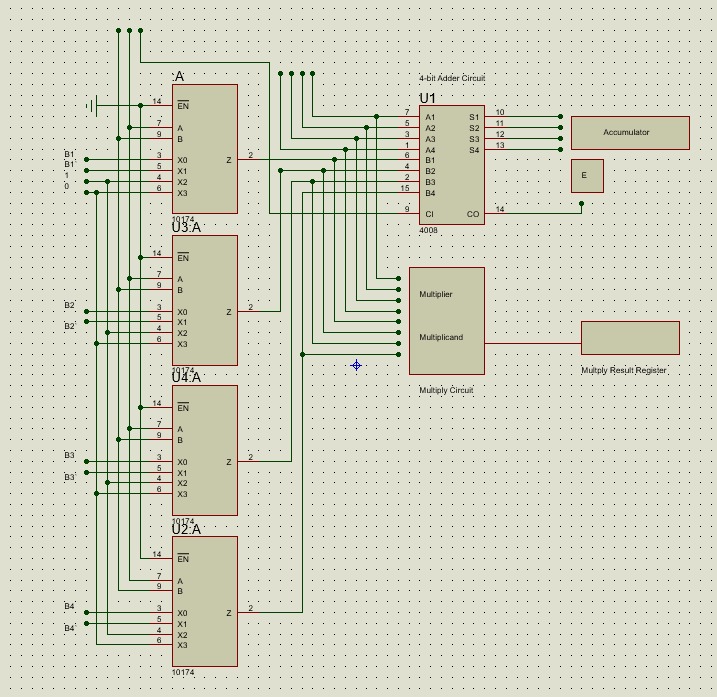
# I/O Instructions:

|  |  |  |
| --- | --- | --- |
| Symbol | Hex Code  MR=00 | Micro-operations  (RC←0 at the end of each) |
| IPT | 0 0 0 1 | AC[0:7]←INP, IFL←0 |
| OTP | 0 0 0 2 | OUT←AC[0:7], OFL←0 |
| ION | 0 0 0 3 | IEN←1 |
| SKO | 0 0 0 4 | If (OFL=0) PC←PC+1 |
| SKI | 0 0 0 5 | If (IFL=0) PC←PC+1 |
| IOF | 0 0 0 6 | IEN←0 |

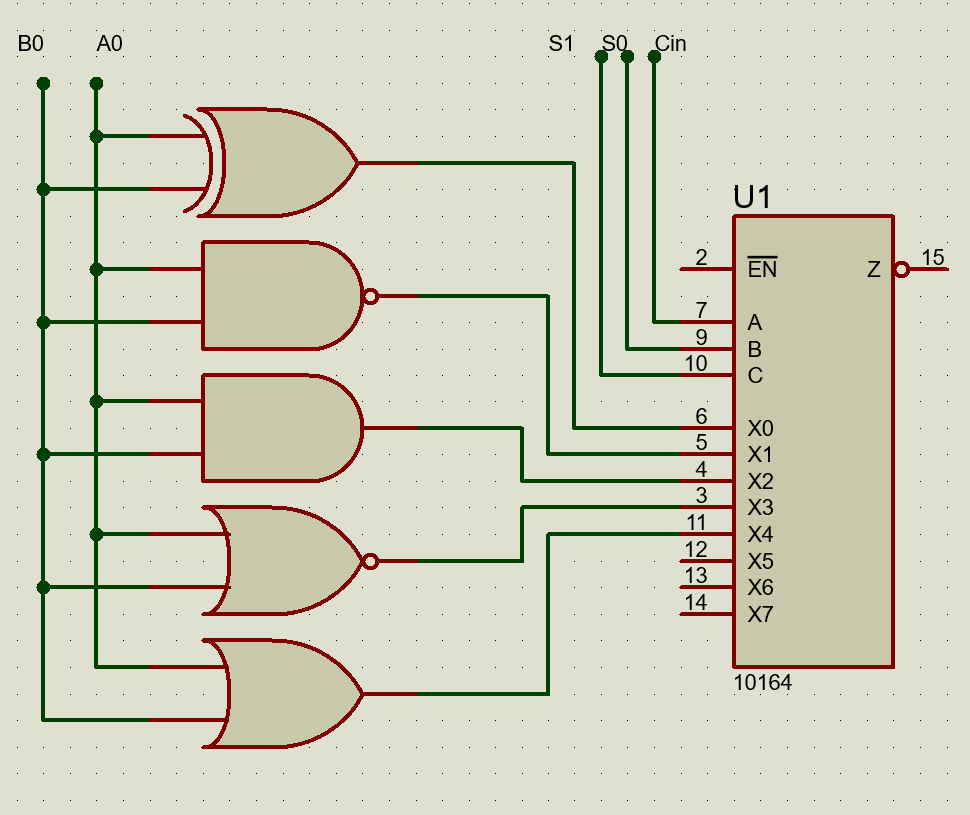
# Instruction Cycle Flowchart:



# Arithmetic Unit:

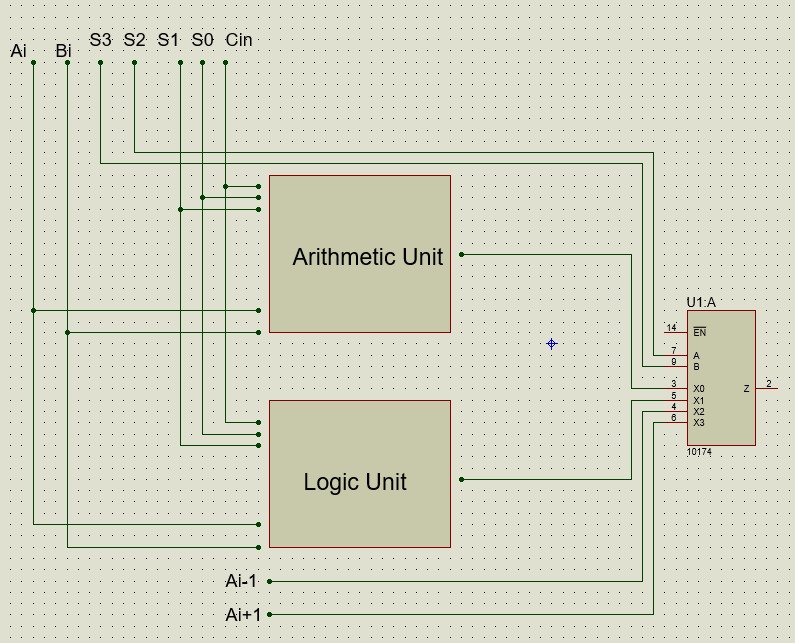


# Logic Unit:



# Arithmetic-Logic-Shift Unit:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| S3 | S2 | S1 | S0 | CIN |  |  |
| 0 | 0 | 0 | 0 | 0 | ADD | A + B |
| 0 | 0 | 0 | 0 | 1 | ADD with Carry | A + B + 1 |
| 0 | 0 | 0 | 1 | 0 | SUB with Borrow | A + B’ |
| 0 | 0 | 0 | 1 | 1 | SUB | A + B’ + 1 |
| 0 | 0 | 1 | 0 | 0 | DEC | A - 1 |
| 0 | 0 | 1 | 0 | 1 | Transfer A | A |
| 0 | 0 | 1 | 1 | 0 | Transfer A | A |
| 0 | 0 | 1 | 1 | 1 | INC | A + 1 |
| 0 | 1 | 0 | 0 | 0 | XOR | A xor B |
| 0 | 1 | 0 | 0 | 1 | NAND | (A ∧ B)’ |
| 0 | 1 | 0 | 1 | 0 | AND | A ∧ B |
| 0 | 1 | 0 | 1 | 1 | NOR | (A ∨ B)’ |
| 0 | 1 | 1 | 0 | 0 | OR | A ∨ B |
| 1 | 0 | x | x | x | SHL | An-1…A0 0 |
| 1 | 1 | x | x | x | SHR | 0 An…A1 |



# Multiply Circuit:

The method for multiplication is as follows:

C4 C3C2

A4  A3  A2  A1

X B4  B3  B2  B1

—----------------------------------------------

      C7 R7 R6 R5  R4  R3 R2  R1

R1 = A1 \* B1

R2 = A2\*B1 + A1\*B2 [1-bit carry = C2] (Max = 1)

R3 = A3\*B1 + A2\*B2 + A1\*B3 + C2 [2-bit carry = C3] (Max = 10)

R4 = A4\*B1 + A3\*B2 + A2\*B3 + A1\*B4 + C3 [2-bit carry = C4] (Max = 11)

R5 = A4\*B2 + A3\*B3 + A2\*B4 + C4 [2-bit carry = C5] (Max = 11)

R6 = A4\*B3 + A3\*B4 + C5 [2-bit carry = C6] (Max = 10)

R7 = A4\*B4 + C6 [1-bit carry = C7] (Max = 1)

Circuit Is designed for 4-bit numbers only but it can be extended for 16-bit operands.

## Example:

Take two numbers 846 and 965. Same can be done both in binary and decimal.

8 4 6

X 9 6 5

------------------------------------------------

C5 R5 R4 R3 R2 R1

R1 = 6\*5 = 30 [C1 = 3] [S1 = 0]

R2 = 5\*4 + 6\*6 + C1 = 59 [C2 = 5] [S2 = 9]

R3 = 5\*8 + 9\*6 + 6\*4 + C2 = 123 [C3=12] [S3 = 3]

R4 = 6\*8 + 9\*4 + C3 = 96 [C4=9] [S4 = 6]

R5 = 9 \* 8 + C4 = 81 [C5=8] [S5 = 1]

Answer = 8 1 6 3 9 0

Diagram, schematic

Description automatically generated

# References:

1. Multiplication method inspired by the Book “Secrets of Mental Math” by Arthur Benjamin and Michael Shermer.