4096 X 17 Computer System Report

by 2022455 .

Submission date: 29-Apr-2024 03:48AM (UTC-0700)
Submission ID: 2365420357
File name: CoalSemesterProject_final-1.pdf (2.96M)
Word count: 4207
Character count: 23088

FCSE

CE222E - Computer Organization & Assembly Language

Instructor: Prof Dr. Ghulam Abbas

4096 X 17 Computer System Report

Group Members:

Yasir Khan (2022455)

Mohammad Bilal Aslam (2022361)

Syed Ahmed Haseeb (2022557)

Aizaz Ur Rehman (2022078)

Date of submission: April 29, 2024

Table of Contents

Abstract

This project details how a basic computer is built and how it carries out its rudimentary functions, which are taking instructions as input, executing the necessary micro-operations, and outputting the result. The instruction is composed of three components: the addressing mode, the op-code, and the address of the operand. This computer is also equipped with a comparator register which increases the number of conditional jump instructions to four. It saves a developer time and hence is used for more thorough assembly language code writing.

Additionally, the computer is equipped with a 16-bit dual-bus architecture that separates the data lines from the control bits. Such approach effectively integrates both the micro-instruction and the control signals using a ring counter to synchronize the various components. The inclusion of a swap instruction can easily swap the values of The Accumulator and B-Register by the programmer. Our basic computer also has all shift operations added to ease the operations of shift operations as only circular shifts were available.

In certain situations, loading an operand by accessing an address takes up unnecessary memory and is complex. To prevent this, a program has been implemented that loads the operand straight into the accumulator.

For additional functionality, we have implemented an AND and NOT on top of the logical OR.

Main memory

A computer's main memory unit is a device that enables quick data storage and retrieval. It keeps data that the CPU (Central Processing Unit) is processing. The data involves the instructions (codes) required for processing as well as the output of these instructions. As seen in Figure 1, the main memory holds 4096 words, each of which is 17 bits long, and the instruction set is broken down into three parts:

- 1 bit represents the address mode,
- 4 bits represent the op-code,
- 12 bits represent the operand.

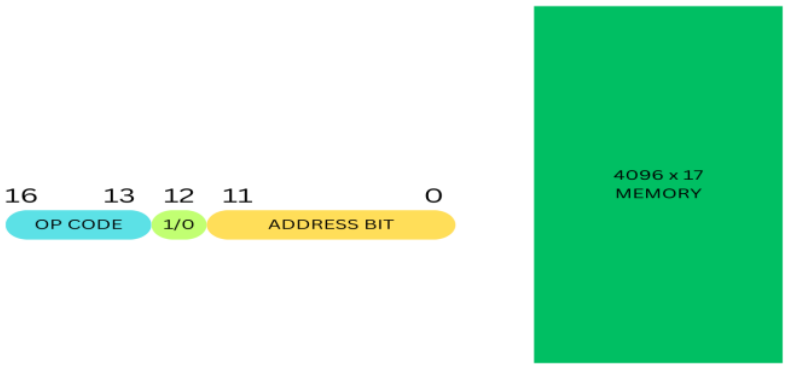


Figure 1

Registers

The CPU contains registers which is a sort of computer memory used to store and manage data while instructions are being executed. Any type of data (such as a bit sequence or individual characters) or an instruction can be stored in a register. These components are represented in figure two and in table one below, with the descriptions of their functions.

Figure 2:

Registers	Symbol	Function
Instruction	IR	saves a word loaded in the memory
Temporary Register	TR	dual short-term register
Program Counter	PC	Save instruction that has to be executed from a memory location
Memory Access Register	M[AR]	location for the fetched instruction is saved
Accumulator	ACCM	Saves all logical and arithmetic details.
F Register	F	saves subsequent operand of arithmetic procedures
Output Register	OPR	saves output value of the code
Input Register	IPR	saves input
Comparison Register	COMP	saves four-bit comparison result

Table 1

Additionally, we have utilized several flip-flops, which are memory devices with two stable states that can be used to store one binary bit for a variety of purposes, such as storing overflows, handling interrupts, and defining addressing modes. Table 2 displays a list of these flip-flops along with the functions they perform.

Flip-Flop	Symbol	Function
Interrupt Enabled	IEN	notifies if the interrupt handling is active
Interrupted	INT	updates if an interruption has taken place
Skip Flag	SKF	updates if to jump to the next instructions or not
Arithmetic Overflow	AOFB	saves a carry of an arithmetic procedure
Shift Overflow	SOFB	saves a carry of a shift procedure
Operational	CURMOD	notifies if the system is active or not
Direct/Indirect	ADDRMOD	shows the addressing mode
Input Flag	INPR	notifies if input is to be expected
Output flag	OUTR	notifies if output is to be expected

Table 2

The Bus System

A dual-bus system is a computer architectural design that uses two different data buses to facilitate communication between various components of the computer system. The system can handle several data transfers at once since it has two data buses. This results in a reduction in congestion and the overall system performance is enhanced. They also make use of parallelism by handling several data transfers to happen at once, which results in reduced latency, and higher throughput. This is the reason we implemented a dual-bus system in our computer as it offers better efficiency and performance than single-bus architectures.

A dual-bus system consists of:

- **Data Bus**: its primary function is to move data between the CPU, memory, and other components and devices.
- **Control Bus:** responsible for managing flow of data and controlling the behavior of the system's components.

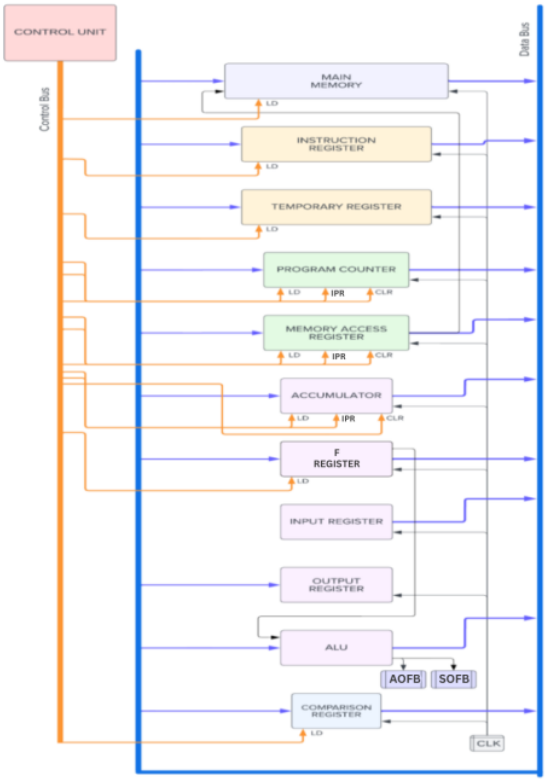


Figure 3

The data bus

A 16-bit data bus is implemented for transporting data between the registers as shown in figure 3. The decoder system picks the register and the load input controls when to let the bus load the data onto the register.

To choose a register and transfer its contents to bus, the decoder combines a 4x10 decoder with three-state gates. Figure 4 is an illustration of how the first bit is chosen. For every bit, the same circuit is duplicated seventeen times.

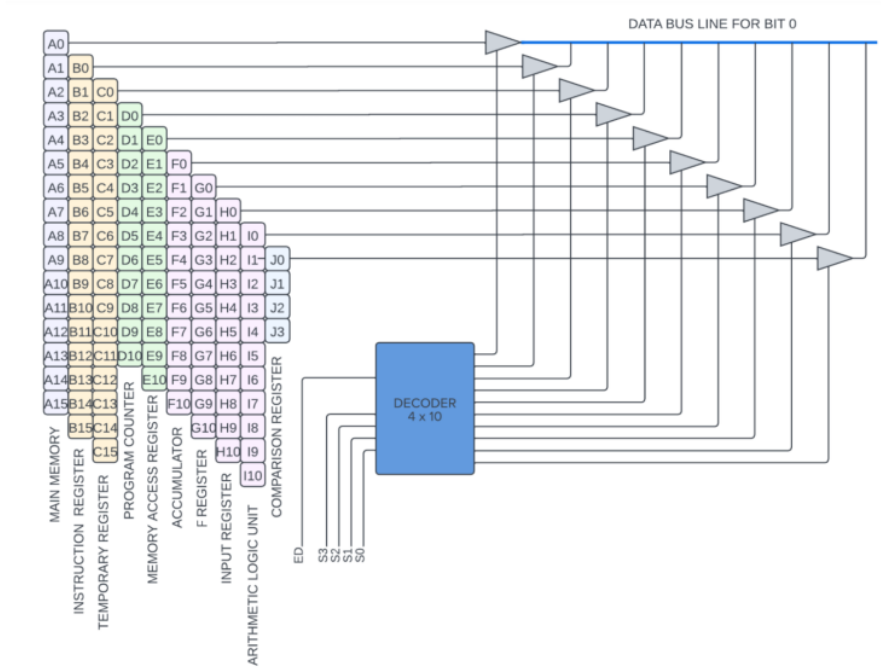


Figure 4

The control unit provides the decoder's four select inputs, which let users choose a particular register. The decoder also receives an additional ED (Enable Decoder) input which works when it is enabled at 1. When required, this input turns on the decoder. Table 4 illustrates how selecting distinct registers is made possible by various select inputs.

E_D	S_3	S_2	S_1	S_0	**Registers**
0	X	X	X	X	None
1	0	0	0	0	Main Memory
1	0	0	0	1	Instruction Register
1	0	0	1	0	Temporary Register
1	0	0	1	1	Program Counter
1	0	1	0	0	Memory Access Register
1	0	1	0	1	Accumulator
1	0	1	1	0	F Register
1	0	1	1	1	Input Register
1	1	0	0	0	Arithmetic Logic Unit
1	1	0	0	1	Comparison Register

Table 3

The control bus

The three control inputs of each register—load, increment, and clear—determine the control of that register. These inputs, which are delivered separately for each register via the control bus, are utilized to control the register operations (Figure 3). To synchronize all register operations, each register receives an additional clock signal.

The control unit can perform any required register action by manipulating a 16-bit control string that is carried via the control bus. The following is the control string's sequence, with Table 4 highlighting bit functions.

Bit	Output Register	Result
L_{MM}	Main Memory	Brings up information of the data bus
L_{IR}	Instruction Register	
L_{TR}	Temporary Register	
L_{PC}	Program Counter	
L_{MAR}	Memory Access Register	
L_{AC}	Accumulator	
L_{F}	F Register	
L_{CR}	Comparison Register	
E_{D}	(Determined by decoder input)	Stores information onto the data bus
I_{PC}	Program Counter	Increment
I_{MAR}	Memory Access Register	
I_{AC}	Accumulator	
C_{PC}	Program Counter	remove information
C_{MAR}	Memory Access Register	
C_{AC}	Accumulator	

Table 4

The Arithmetic Logic Unit (ALU)

An arithmetic logic unit is a digital circuit responsible for performing arithmetic, logical, and shift operations. It is given two inputs and runs a sequence of instructions. After that, the result is stored in a register or memory location. It is a fundamental component of a CPU. The accumulator provides the primary operand C, whereas the E register provides the secondary operand.

The ALU is made up of two parts: the arithmetic and logic unit. These two can be combined to give rise to a Shift unit and a Comparator unit.

Arithmetic Unit

The arithmetic unit can perform either the addition or subtraction of binary bits. This is done by the binary adder/subtractor combinational circuit (Figure 5) that makes up the arithmetic unit. It has a carry input in addition to a select input for selecting which operation to execute. Any excess flow generated is kept in the AOFB Flip-flop. Table 5 illustrates the specific arithmetic operations that can be carried out using the different select inputs.

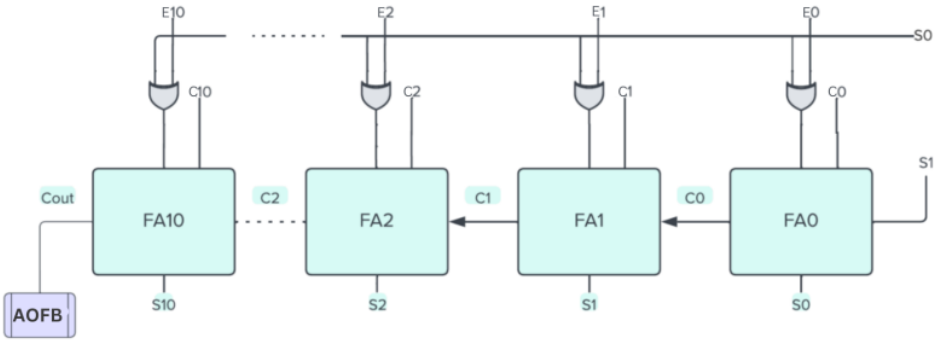


Figure 5

S_1	S_0	Function
0	0	Add
0	1	Subtract with borrow
1	0	Add with Carry
1	1	Subtract

Table 5

Logic Unit

Logical operations such as AND, OR, NOT, NAND, NOR, XOR, and XNOR are carried out on binary values by the logic unit (Figure 6). In order to accomplish this, it combines multiplexers and de-multiplexers, using select inputs to determine which logical operation to carry out. It has bitwise operations capabilities as well. Table 6 illustrates the specific logical operations that can be carried out with the different select inputs.

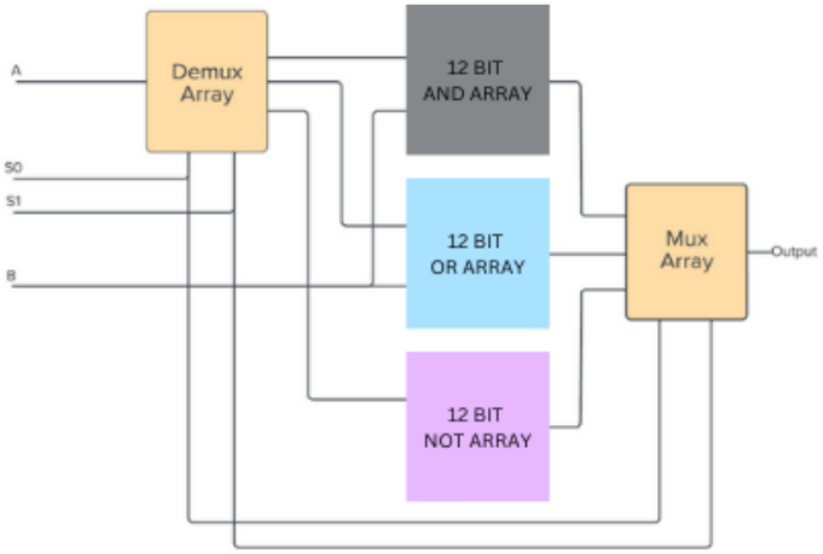


Figure 6

S_1	S_0	Function
0	0	C ∧ E
0	1	C ∨ E
1	0	C’

Table 6

Shift Unit

The shift units primary concern is to perform shift operations on data. The bits of a binary number are moved left or right during these operations. We execute shfr, shfl, cirr, cirl, atshr, and atshl in our shift unit. The arithmetic unit and the logic unit are combined to create the shift unit.

This results in two different kinds of shift operations: logical shift operations, where the shift operation creates empty bit positions that are filled with zeros to maintain the data's sign. To maintain the sign and magnitude of the data, empty bit positions in arithmetic shift operations are filled using the original data's sign bit. Table seven describes the truth table of the shift unit.

S_2	S_1	S_0	Function
0	0	0	SHFR
0	0	1	SHFL
0	1	0	CIRR
0	1	1	CIRL
1	0	0	ATSHR
1	0	1	ATSHL

Table 7

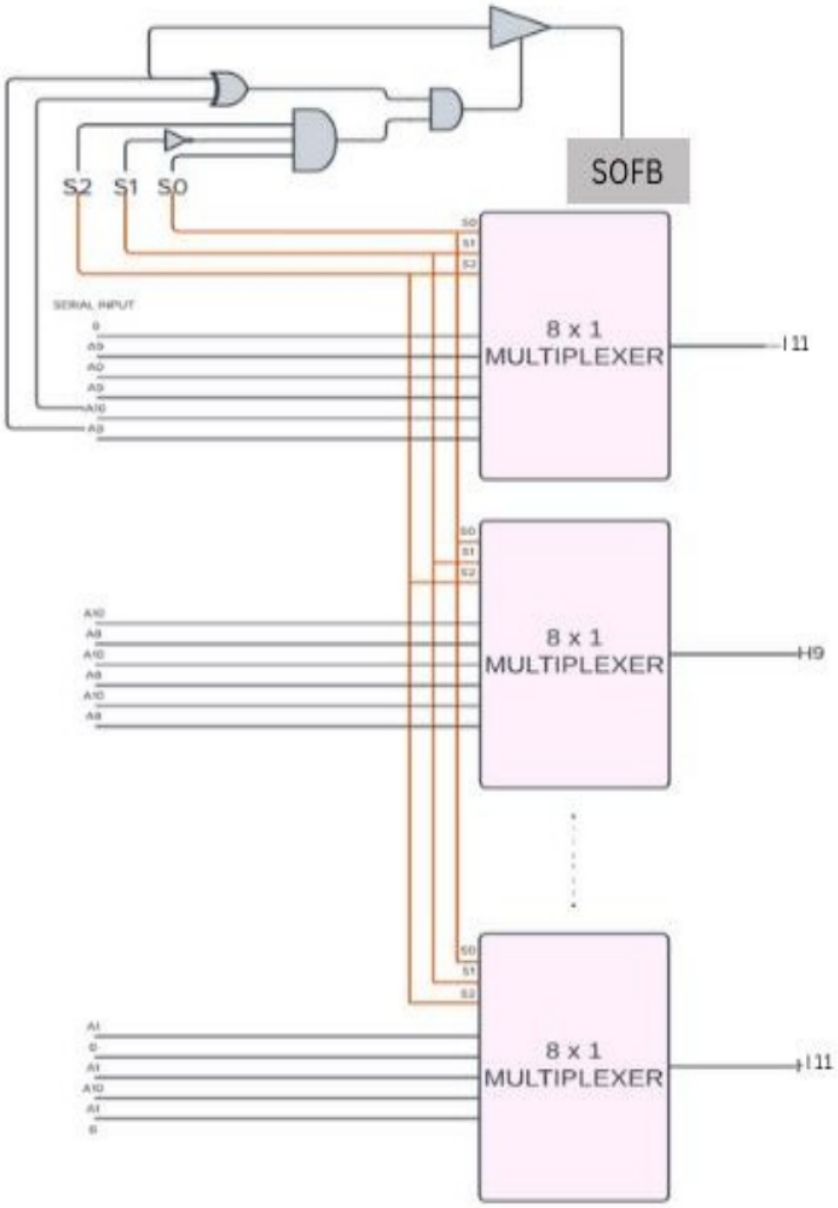

SOFB
S2
S1
S0
8 x 1
MULTIPLEXER
I11
8 x 1
MULTIPLEXER
H9
8 x 1
MULTIPLEXER
I11

Figure 7

Comparator Unit

A comparator unit compares two binary numbers and decides if they are equal or if one is greater or less than the other.

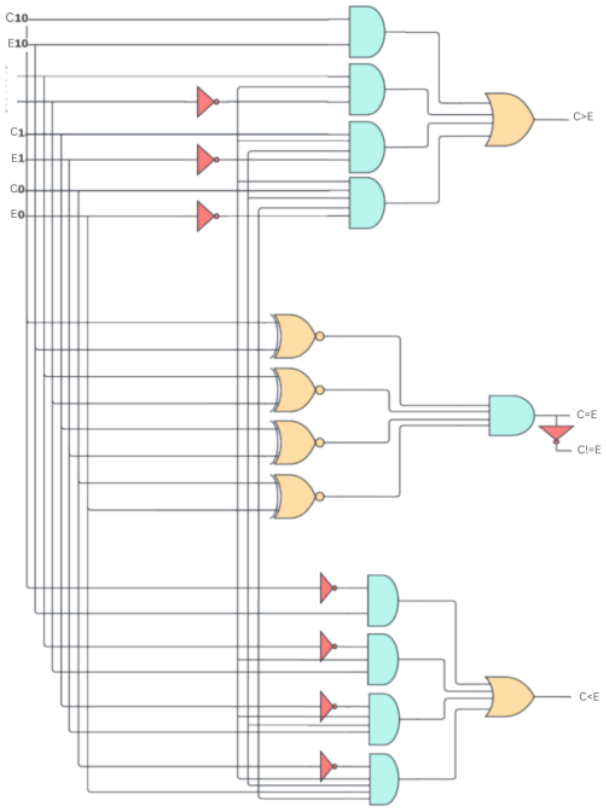


Figure 8: The Comparator Unit

Comparison	Output
C >E	1000
C < E	0100
C = E	0010
C ≠ E	0001

Table 8

ALU

Figure 9 illustrates how an ALU's system integrates all of these many parts. The ALU utilizes five select bits total, two of which are used to pick the component (S3 and S4) and three of which are used to select the operation (S0, S1, and S2). Table 9 demonstrates the use of a mux and de-mux array in determining which of the ALU's components is selected.

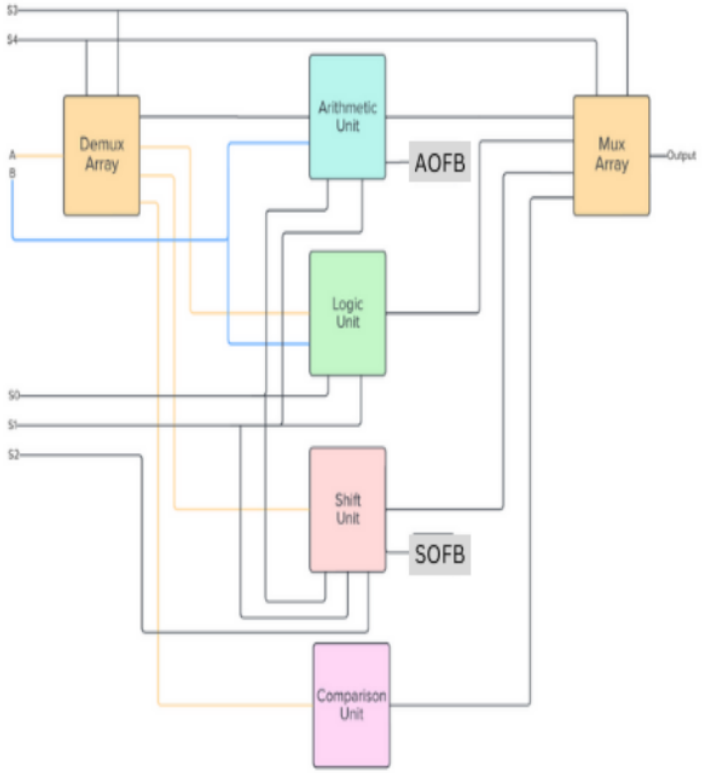


Figure 9: The ALU

S1	S0	Selected Component
0	0	Arithmetic component
0	1	Logic component
1	0	Shift component
1	1	Comparator component

Table 9

Instruction Set Architecture (ISA)

An instruction set architecture is a model of a computer system capable of comprehending and executing instructions. Three categories of instructions are defined by the ISA: branch and jump, data transfer, and arithmetic/logic instructions. An instruction set is made up of the addressing mode, op-code, and the operand's address as previously mentioned. A 17-bit instruction is divided into 4 bits to indicate the kind of instruction and 1 bit to indicate the instruction's addressing mode. As a result, 32 distinct commands in all are made possible which are broken down into two primary groups:

- 10 register-operation (direct or indirect) instructions
- 12 utility instructions

Addressing Modes

An addressing mode is a technique to specify how operands for an instruction are accessed. An instruction's operands can be found and retrieved depending on the addressing method used; directly or indirectly. The operand's location that the operation is to performed on is contained in the operand section of a direct instruction. On the other hand, the operand part of an indirect instruction stores the operand's location whose location stores the operand that has to be operated on.

Register-Operation Instructions (Op-code and address of operand)

Instructions in an ISA that carry out operations directly on CPU registers are known as register operation instructions. The location where the instruction is to be executed is stored in them. Table 10 lists the register-operation instructions, their mnemonics, and their functions.

Op-code	Mnemonic	Function
0000	LDA	Load location to ACCM
0001	STA	saves ACCM to address
0010	ADD	Add to ACCM
0011	SUB	Subtract from ACCM
0100	CMP	Compare with ACCM
0101	JMP	Jump to location
0110	LAND	AND operation

0111	LOR	OR operation
1000	LNOT	NOT operation
1001	SWP	Switch ACCM with location contents

Table 10

CMP

Detailed description of the compare instruction. It looks for four comparisons and stores the value in the comparison register as shown in Table 11.

Comparison	**Comparison registers value**
AC > operand	1000
AC < operand	0100
AC = operand	0010
AC ≠ operand	0001

Table 11

Additional Instructions

An addressing bit can be used as an extra bit in the op-code to carry out more instructions. Moreover, a lot of instructions use the operand portion of the instruction to further define the instruction's mode. A shift instruction, for instance, indicates the kind of shift that should be made in the operand of the instruction. Table 12 lists these instructions, mnemonics, and their functions.

Op-code	Addressing bit	Mnemonic	Function
1010	0	LDI	load the immediate operand to accumulator
1011	0	INC	increment
1100	0	CLR	clear the register
1101	0	SHFT	perform shift operation
1110	0	IEN	enable/disable Interrupt
1111	0	BSA	branch and save the current location to the given location
1010	1	SKP	skip instruction if the skip flag is on
1011	1	JMPL	if less, jump to specified location
1100	1	JMPE	if equal, jump to the specified location
1101	1	JMPN	if not equal, jump to the specified location
1110	1	JMPG	if greater, jump to the specified location
1111	1	END	stop the operations

Table 12

1) **LDI**
This is implemented to load the content directly onto the accumulator.

2) **INC**
This is utilized either to increment the content of a register or to enable a flip-flop. The operand decides whether a flip-flop or register is chosen.

Binary	Hex	Register or Flip-flop
0000 0000 0001	001	Accumulator
0000 0000 0010	002	F Register
0000 0000 0100	004	IEN
0000 0000 1000	008	SKF
0000 0001 0000	010	INPR
0000 0010 0000	020	OUTR

Table 13

3) **CLR**
This is utilized to clear a registers contents, or to disable a flip-flop. Table 14 describes these functions.

Binary	Hex	Register or Flip-flop
0000 0000 0001	001	Accumulator
0000 0000 0010	002	F Register
0000 0000 0100	004	COMP
0000 0000 1000	008	IEN
0000 0001 0000	010	SKF
0000 0010 0000	020	INPR
0000 0100 0000	040	OUTR
0000 1000 0000	080	AOFB
0001 0000 0000	100	SOFB

Table 14

4) **SHFT**

With the shift, it is used to obtain the desired shift operation on the contents of the accumulator. The desired shift is chosen according to the value of the operand.

Binary	Hex	Operation
0000 0000 0001	001	Logical Left Shift
0000 0000 0010	002	Logical Right Shift
0000 0000 0100	004	Circular Left Shift
0000 0000 1000	008	Circular Right Shift
0000 0001 0000	010	Arithmetic Left Shift
0000 0010 0000	020	Arithmetic Right Shift

Table 15

5) **IEN**

The IEN instruction activates the interrupt flip-flop IEN based on the value of the operand.

Operand (Binary)	Operand (Hex)	IEN state
0000 0000 0001	001	Enable
0000 0000 0010	002	Disable

Table 16

6) **BSA**

BSA is the location where you jump to and store the location of the program counter as the return address. This permits the system to perform and handle subroutines.

7) **SKP**

The SKP instruction goes to the following instruction if the skip flag is active.

8) **JMPL**

The JMPL instruction jumps to a location if the comparison register result was 'less than', which corresponds to 0100.

9) **JMPE**

The JMPE instruction jumps to a location if the comparison register result was 'equal to', which is the value 0010.

10) **JMPN**

The JMPN instruction jumps to a location if the comparison register result was 'not equal to', which corresponds to 0001.

11) **JMPG**

The JMPG instruction jumps to a location if the comparison register result was 'greater than', which corresponds to 1000.

12) **END**

The END instruction notifies the end of the program. It stops the operations, resets the value of PC to 0, and disables the OUTR flip-flop.

The Control Unit (CU)

A key part of a computer's central processing unit (CPU) is the control unit, which interprets user input and converts it into control signals for operations. It coordinates data transfers, exchanges data with the arithmetic logic unit (ALU), and manages the data flow (I/O) between the central processing unit (CPU) and external devices. The control unit's primary function comprises fetching, decoding, and executing instructions. The three parts that make up the control unit are explained below.

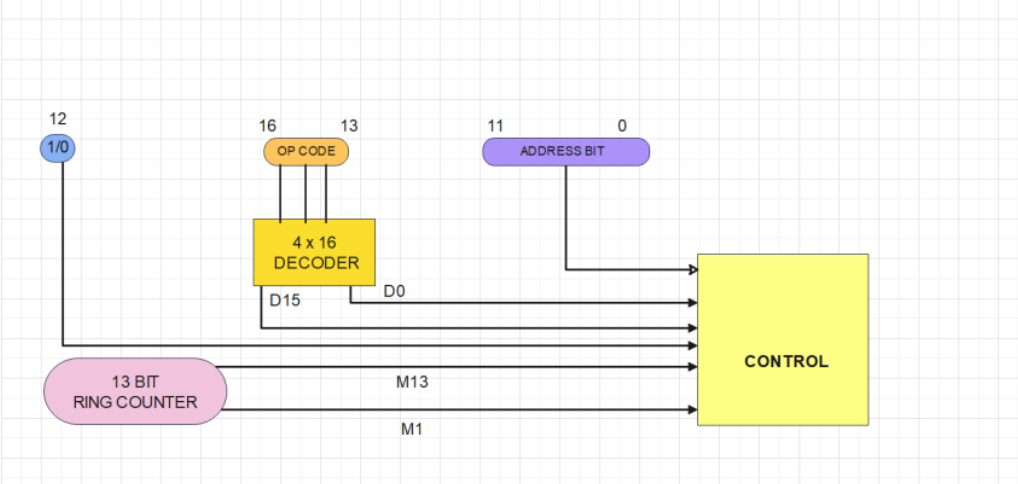


Figure 10

Instruction Decoder

An instruction decoder's job is to convert an instruction into its individual micro-operations. To locate the address where the micro-operations will start, the instruction is split down into its addressing mode, op-code, and operand address.

Ring Counter

This is a shift register which is made up of flip-flops made by feeding the output of the final flip-flop to the input of the first creating a ring structure. It's a 13-bit counter which determines what micro-operation is being executed. We used a ring counter over other counters since it can output to 13 lines directly and doesn't need a second decoder to split a binary number into various outputs.

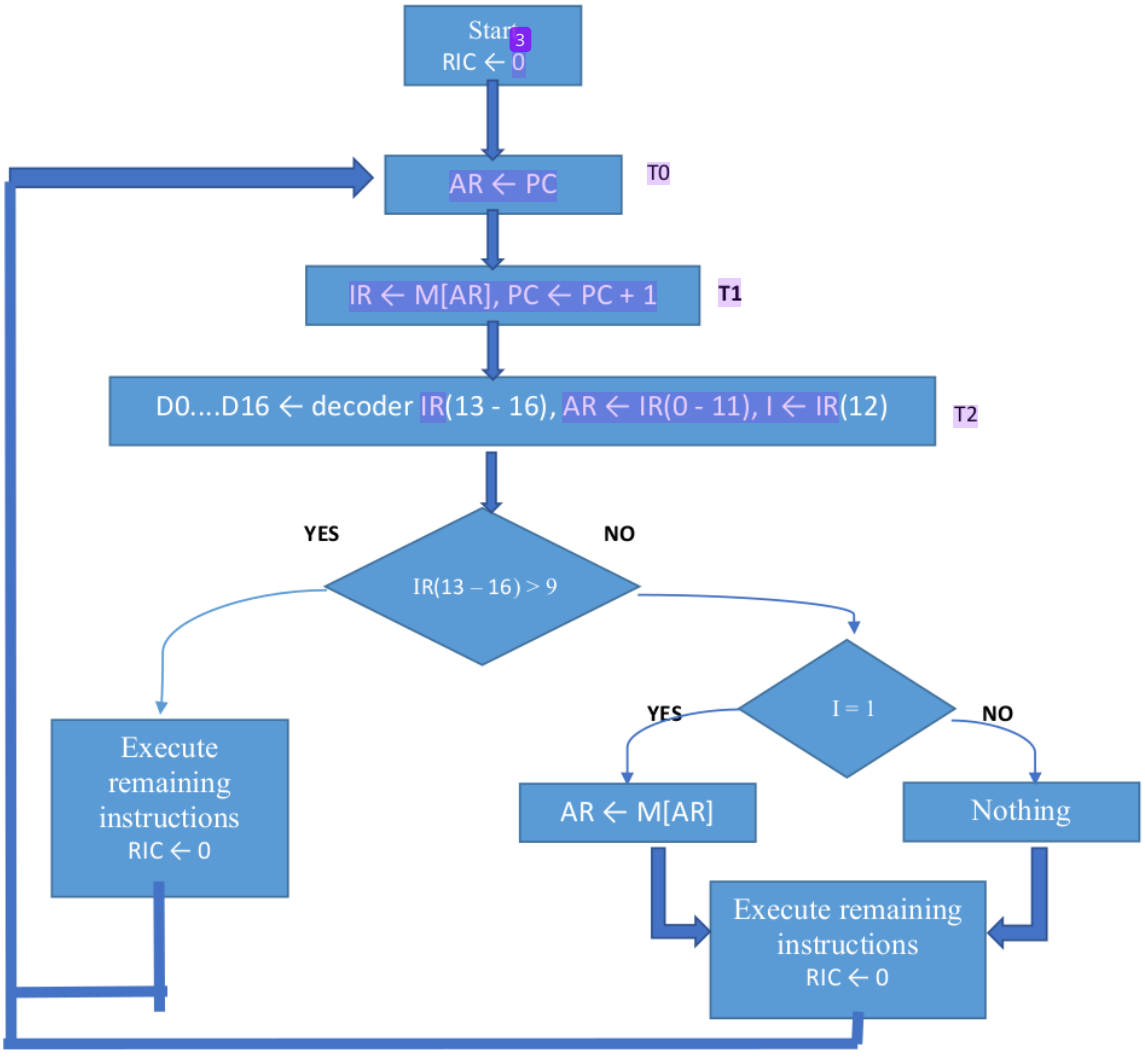
Control Logic

The control logic is responsible for directing the operation of the system. It responds to the commands entered by the user and software. It provides the control signals required to synchronize the execution of a micro-operation for every input that the instruction decoder and ring counter may receive.

Micro-operations

A micro-operation is the smallest unit of operation performed on the information stored in registers to implement machine instructions. A micro-operation includes three crucial steps: the fetched, decode, and execute cycle. All of these are executed during a single clock cycle.

The ring counter offers thirteen outputs and determines which of these micro-operations is being executed. The fetch instructions are the first two micro-operations (T0, T1). They are succeeded by a decode micro-operation (T2). If the addressing mode is indirect, an extra micro-operation (T3) is carried out. The fetch and decode stages of an instruction's execution consist of these four basic micro-operations, which are shared by all instructions. Each of the remaining instructions (T4 through T12) is distinct from the others. The ring counter is reset to 0 to begin processing the micro-instructions for the subsequent instruction after the micro-instructions for the current instruction have been completed. Below is a flowchart detailing the flow of instructions.



1) Fetch

In the fetch micro-operations, the instructions are retrieved from main memory and loaded onto the registers. In T0, the program counter is stored in the access register. In T1, the memory address register is loaded into the instruction register and the program counter is incremented. The micro-operations associated with this are as follows:

$$T_0: AR \leftarrow PC$$

$$T_1: IR \leftarrow M[AR], PC \leftarrow PC + 1$$

2) Decode

The decode micro-operation divides an instruction into three parts: op-code, operand, and addressing mode. The operand is indicated by the first 12 bits, the op-code is indicated by the next four, and the addressing mode is indicated by the last bit, arranged from right to left. An instruction is decoded using the subsequent micro-operation:

T_2: D0....D16 ← decoder IR(13 - 16), AR ← IR(0 - 11), I ← IR(12)

3) Execute

The stage where the machine performs the actual function that the instruction specifies is known as the execute phase of a micro-operation. T4, the first micro-operation is implemented only if the addressing mode is indirect. The micro-operation associated with this is as follows:

$$D_7 T_3: AR \leftarrow M[AR]$$

The rest of the micro-operations (T_4 - T_{12}) are indicated as:

LDA	T_5: ACCM ← M[AR], RIC ← 0
STA	T_5: M[AR] ← ACCM, RIC ← 0

ADD	T_5: E ← M[AR] T_6: ACCM ← AC + E AOFB ← Cout, RIC ← 0
SUB	T_5: E ← M[AR] T_6: ACCM ← AC - E, RIC ← 0
CMP	T_5: E ← M[AR] T_6: If (ACCM > E) then COMP ← 8 T_7: If (ACCM < E) then COMP ← 4 T_8: If (ACCM = E) then COMP ← 2 T_9: If (ACCM ≠ E) then COMP ← 1, RIC ← 0
JMP	T_5: PC ← M[AR], RIC ← 0
LAND	T_5: E ← M[AR] T_6: ACCM ← AC ∧ E, RIC ← 0
LOR	T_5: E ← M[AR] T_6: ACCM ← AC $\vee$ E, RIC← 0
LNOT	T_5: ACCM ← M[AR] T_6: ACCM ← ACCM', RIC ← 0
SWP	T_5: TR ← M[AR] T_6: M[AR] ← ACCM T_7: ACCM ← TR, RIC ← 0
LDI	T_5: ACCM ← M[AR], RIC ← 0

INC	T_5: If (M[AR] = 001) then ACCM ← ACCM + 1 T_6: If (M[AR] = 002) then E ← E + 1 T_7: If (M[AR] = 004) then IEN ← 1 T_8: If (M[AR] = 008) then SKF ← 1 T_9: If (M[AR] = 010) then INPR ← 1 T_{10}: If (M[AR]= 020) then OUTR ← 1, RIC ← 0
CLR	T_5: If (M[AR] = 001) then ACCM ← 0 T_6: If (M[AR] = 002) then E ← 0 T_7: If (M[AR] = 004) then COMP ← 0 T_8: If (M[AR] = 008) then IEN ← 0 T_9: If (M[AR] = 010) then SKF ← 0 T_{10}: If (M[AR] = 020) then INPR ← 0 T_{11}: If (M[AR] = 040) then OUTR ← 0 T_{12}: If (M[AR] = 080) then AOFB ← 0 T_{13}: If (M[AR] = 100) then SOFB ← 0, RIC ← 0
SHFT	T_5: If (M[AR] = 001) then ACCM ← shfl ACCM T_6: If (M[AR] = 002) then ACCM ← shfr ACCM T_7: If (M[AR] = 004) then ACCM ← cirl ACCM T_8: If (M[AR] = 008) then ACCM ← cirr ACCM T_9: If (M[AR] = 010) then ACCM ← atshl ACCM T_{10}: If (M[AR] = 020) then ACCM ← atshr ACCM, RIC ← 0
IEN	T_5: If (M[AR] = 001) then IEN ← 1 T_6: If (M[AR] = 002) then IEN← 0, RIC ← 0
BSA	T_5: M[AR] ← PC T_6: AR ← AR+ 1 T_7: PC ← AR, RIC ← 0

SKP	T_5: If (SKF = 1) then PC ←PC+ 1, RIC ← 0
JMPL	T_5: If (COMP = 4) then PC ← M[AR], RIC ← 0
JMPE	T_5: If (COMP = 2) then PC ← M[AR], RIC ← 0
JMPN	T_5: If (COMP = 1) then PC ← M[AR], RIC ← 0
JMPG	T_5: If (COMP = 8) then PC ← M[AR], RIC ← 0
END	T_5: CURMOD ← 0, RIC ← 0

The Interrupt cycle

The last cycle is the interrupt cycle, which is a series of micro-operations that following the execution cycle. Interrupts are signals from the I/O devices to the CPU to solicit a specific event that needs immediate attention. The method through which the CPU responds to the interruptions coming from an external device is an Interrupt Cycle. If the CPU recognizes that an interrupt is happening, it will stop whatever that is currently working on, proceeds to the interrupt, fixes it, and then returns to the former operation**.**

The interrupt at the IEN instruction may be triggered or disabled by presenting operand 001 to activate the IEN flip-op and operand 010 to deactivate the IEN flip-flop . The INPR and INT is 1 for any external input the computer receives in the form of interrupt, which initiate the computer to jump to & run the subroutine to process the interrupt in the following instruction cycle. The subroutine will be kept at memory location 1701–2047, while the memory address 1700 will maintain the original address of the previously executed instructions.

Whenever a program wants to generate an external output while it is running, it sets the output register (OUTR) and interrupt flip-flop (INT) to 1 and then jumps to a subroutine to handle the interrupt.

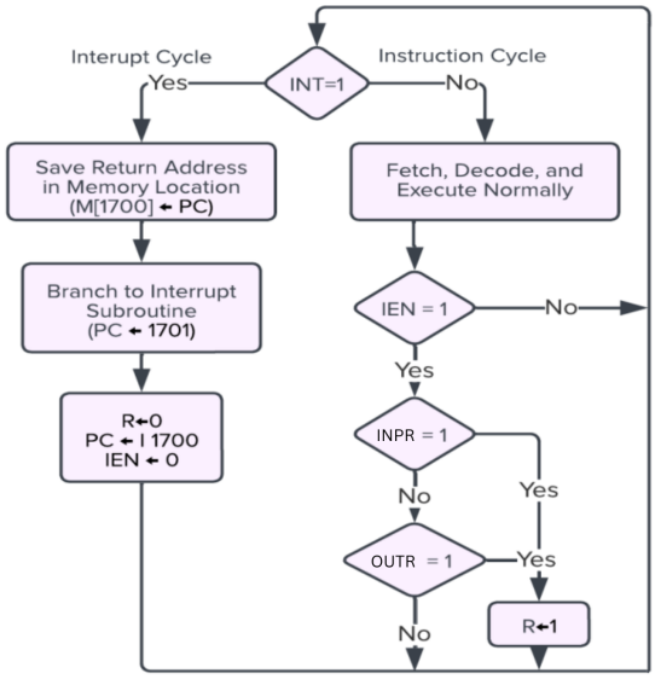


Figure 11

Complete Computer Flowchart

The complete computer flowchart which includes both the interrupt and instruction cycle.

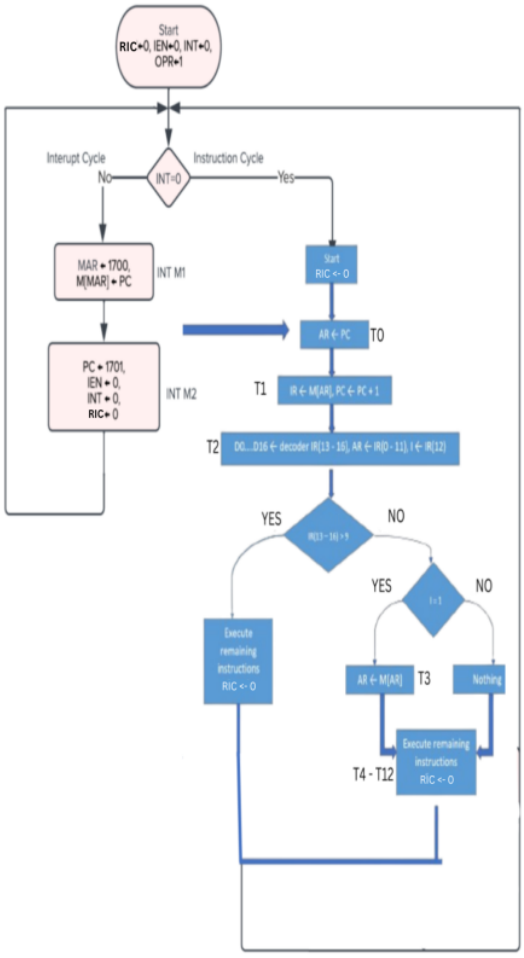


Figure 12

Bibliography

- Bus system, https://www.gatevidyalay.com/system-bus-in-computer-architecture/
- Advantages of a dual bus system, https://smallbusiness.chron.com/benefits-using-multiplebus-architecture-compared-singlebus-architecture-72173.html
- AS & A Level Computer Science 9608: Processor Fundamentals", https://www.cambridgeinternational.org/Images/502960-2021-syllabus.pdf
- Registers, https://www.javatpoint.com/computer-registers
- ALSU, https://medium.com/@suyog.kharche/arithmetic-logic-shift-unit-a6a47c8ba517
- Ring counter, https://www.electrical4u.com/ring-counter/
- ISA, https://www.lenovo.com/us/en/glossary/instruction-set-architecture/
- ARM Instruction Set, https://iitd-plos.github.io/col718/ref/arm-instructionset.pdf
- Magnitude Comparator in Digital Logic, https://www.geeksforgeeks.org/magnitude-comparator-in-digital-logic/
- Implementation of 4-Bit Magnitude Comparator, https://www.deldsim.com/study/material/15/implementation-of-4-bit-magnitude-comparat or-using-ic-74ls85/

- Computer Organization and Architecture, Eleventh Edition, William Stallings, 2022.

- Computer System Architecture, Third Edition, M. Morris Mano, 2007.

4096 X 17 Computer System Report

ORIGINALITY REPORT

16%	3%	0%	15%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Higher Education Commission Pakistan Student Paper	12%
2	Submitted to University College Birmingham Student Paper	1%
3	www.coursehero.com Internet Source	1%
4	Submitted to Middlesex University Student Paper	<1%
5	www.ijert.org Internet Source	<1%
6	Submitted to University of Ghana Student Paper	<1%
7	doczz.net Internet Source	<1%
8	pt.scribd.com Internet Source	<1%
9	B. I. Dahn, H. Wolter. "ORDERED FIELDS WITH SEVERAL EXPONENTIAL FUNCTIONS",	<1%

	Mathematical Logic Quarterly, 2006 Publication	
10	www.holtek.com Internet Source	<1%
11	www.sdsc.edu Internet Source	<1%
12	z3d9b7u8.stackpathcdn.com Internet Source	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		