DT008/2 MICROCOMPUTER SYSTEMS 1 NOTES

CONTENTS

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

SECTION 1

An Introduction to Microprocessors and Microcomputers

A ***microprocessor*** is a large-scale integrated circuit (I.C) device that has the ability to perform tasks in response to a set number of instructions. These instructions are in binary form and are stored in program memory and a sequence of instructions which perform a certain task are called a program. The microprocessor is constructed such that it takes actions on binary data such as addition, multiplication, division, etc in response to the instruction that it is currently executing. The microprocessor operates on a fetch and execute cycle in that it first fetches from program memory the instruction (by reading the contents of the addressed memory location) and then presents this instruction to its internal instruction decoder for execution. This execution could involve any number of tasks such as fetching data from memory, sending data to memory or arithmetic operations on data stored internally in registers called accumulators.

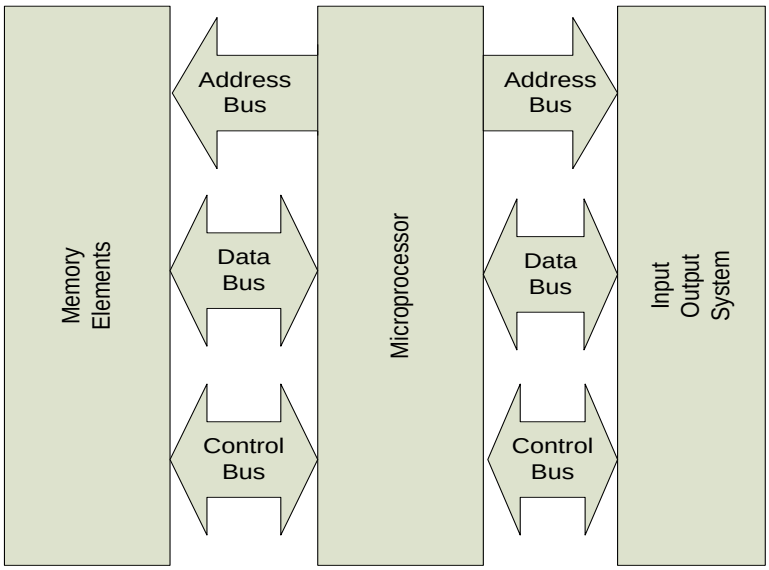
A ***microcomputer*** system consists of a microprocessor, memory to store programs and data and an input output (I/O) system to allow data to be input and output.

In the most basic system the memory can consist of a minimum quantity of read only memory (ROM) to permanently store the program and random access memory (RAM) to store results (data) and the I/O system can consist of a binary switch and a light emitting diode (LED).

More advanced systems could have RAM, ROM and erasable programmable ROM (EPROM) for development purposes as well as a keyboard and a display system.

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

Block Diagram of a microcomputer system



The diagram above shows the three main elements of a microcomputer system which are

Memory,

Microprocessor (or CPU Central Processing Unit) and

Input/Output (I/O) system.

The microprocessor is connected to each of the other two elements by means of circuit board tracks or wires which are grouped together according to their common functions into three buses: the address bus, the data bus and the control bus.

The address bus contains all the output lines of the microprocessor required for addressing a memory location. The address bus is unidirectional since the microprocessor only outputs addresses.

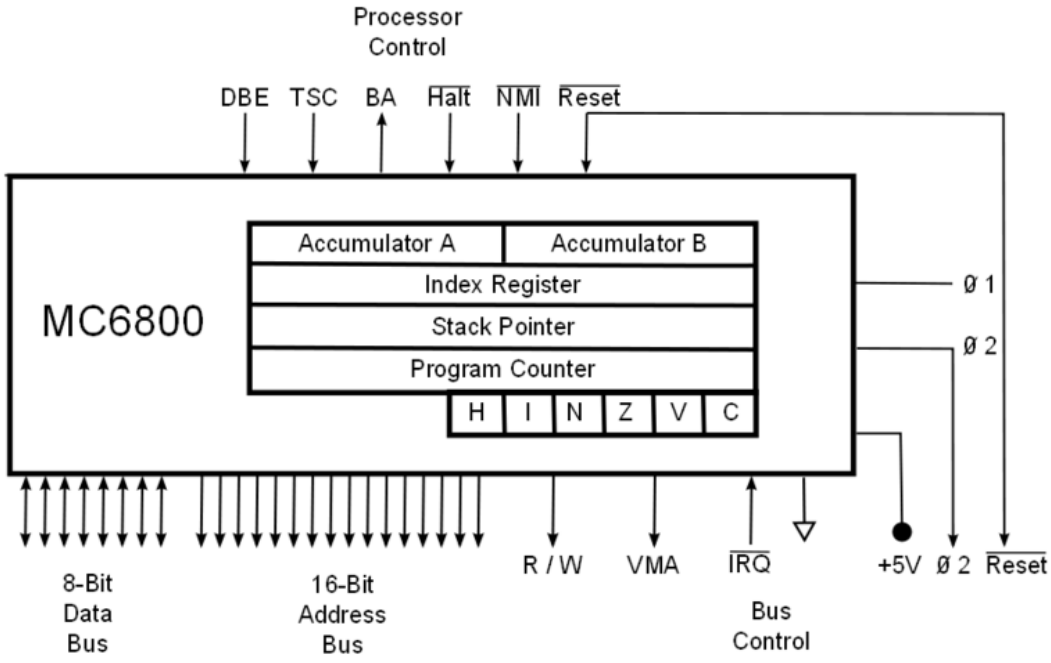
The data bus contains all the microprocessor lines required for the transfer of data to/from a memory element or a dedicated I/O device. This bus is bidirectional since data can flow to or from all devices.

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

The control bus contains all of the microprocessor lines required for the control of the microprocessor and the elements connected to it such as for example: read/write lines, enable lines, interrupt lines etc. This is also a bidirectional bus since the microprocessor requires both input and output signals.

Block diagram of internal registers and pin out of 6800 microprocessor

Below is a block diagram of a Motorola 6800 8 bit microprocessor showing the internal registers and the pins associated with the three buses described above.



This microprocessor has the following internal registers:

Two 8 bit general purpose accumulators A and B (Acc (A), Acc (B) or often just A and B) which are used for temporarily storing the results of arithmetic operations,

A 16 bit Index Register (IX) which is used with the indexed addressing mode to point to an address in memory,

A 16 bit Stack Pointer (SP) which points to an area of memory, called the stack, set aside for data storage during program execution,

A 16 bit Program Counter (PC) which always points to the next instruction address and is automatically incremented as part of the fetch execute cycle and,

An 8 bit Condition Code Register (CCR) of which only 6 bits are used. This is used to indicate the arithmetic results of operations.

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

Most if not all microprocessors will have these and more internal registers often with different names but with the same function.

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

SECTION 2

Revision of Number Systems

The Binary, Hexadecimal and Two's Complement number systems.

The Binary Number System

The 8 bit binary number system uses 8 binary bits (0 or 1) to represent decimal numbers in the range 0 – 255.

The diagram below shows the decimal value according to where a 1 is within the binary number.

128	64	32	16	8	4	2	1

For example the binary number 01101101 is converted to decimal by addition of the numbers that the ones represent in the table:

128	64	32	16	8	4	2	1
0	1	1	0	1	1	0	1

$01101101_2 = 64+32+8+4+1 = 109_{10}$

The subscript 2 represents the binary number system (base 2) and the decimal number system has subscript 10 (base 10).

By a similar process the decimal number 145 can be converted to binary by breaking it down into an addition of the numbers in the table:

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

$145_{10} = 128_{10} + 16_{10} + 1_{10} = 10010001_2$

128	64	32	16	8	4	2	1
1	0	0	1	0	0	0	1

The Hexadecimal Number System

The hexadecimal (hex) number system uses a hex digit to represent a combination of 4 binary bits according to the table below:

4 BIT BINARY	DECIMAL VALUE	HEXADECIMAL DIGIT
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Hence an 8 bit binary word is represented by two hex digits for example

$11001010_2 = CA_{16}$ (= 202_{10}) Note the subscript for hex (base 16 since there are 16 hex digits).

Note: $00000001_2 = 01_{16} = 01_{10}$

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

The Twos Complement Number System

With the 8 bit binary number system decimal numbers in the range 0-255 can be represented where 00000000 = 0 and 11111111 = 255

With the twos complement number system both positive and negative numbers can be represented where the most significant bit (MSB) determines the sign – MSB = 0 means a positive number and MSB = 1 means negative.

The table below shows the range of 8 bit twos complement numbers:

Twos complement	Decimal Number	Hexadecimal
01111111	+127	7F
01111101	+126	7E
01111100	+125	7D
00000010	+2	02
00000001	+1	01
00000000	0	00
11111111	-1	FF
11111110	-2	FE
10000001	-127	81
10000000	-128	80

Thus the range of 8 bit twos complement numbers is

-127 to +128

And in **HEXADECIMAL** the range is

80 to 7F

To convert an 8 bit binary number into twos complement form first invert all the bits (0 becomes 1 and 1 becomes 0) and the add 1.

For Example

00000001 = +1

Inverting 00000001 becomes 11111110

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

Adding 1 11111110

+1

Answer 11111111 (ignore any carry)

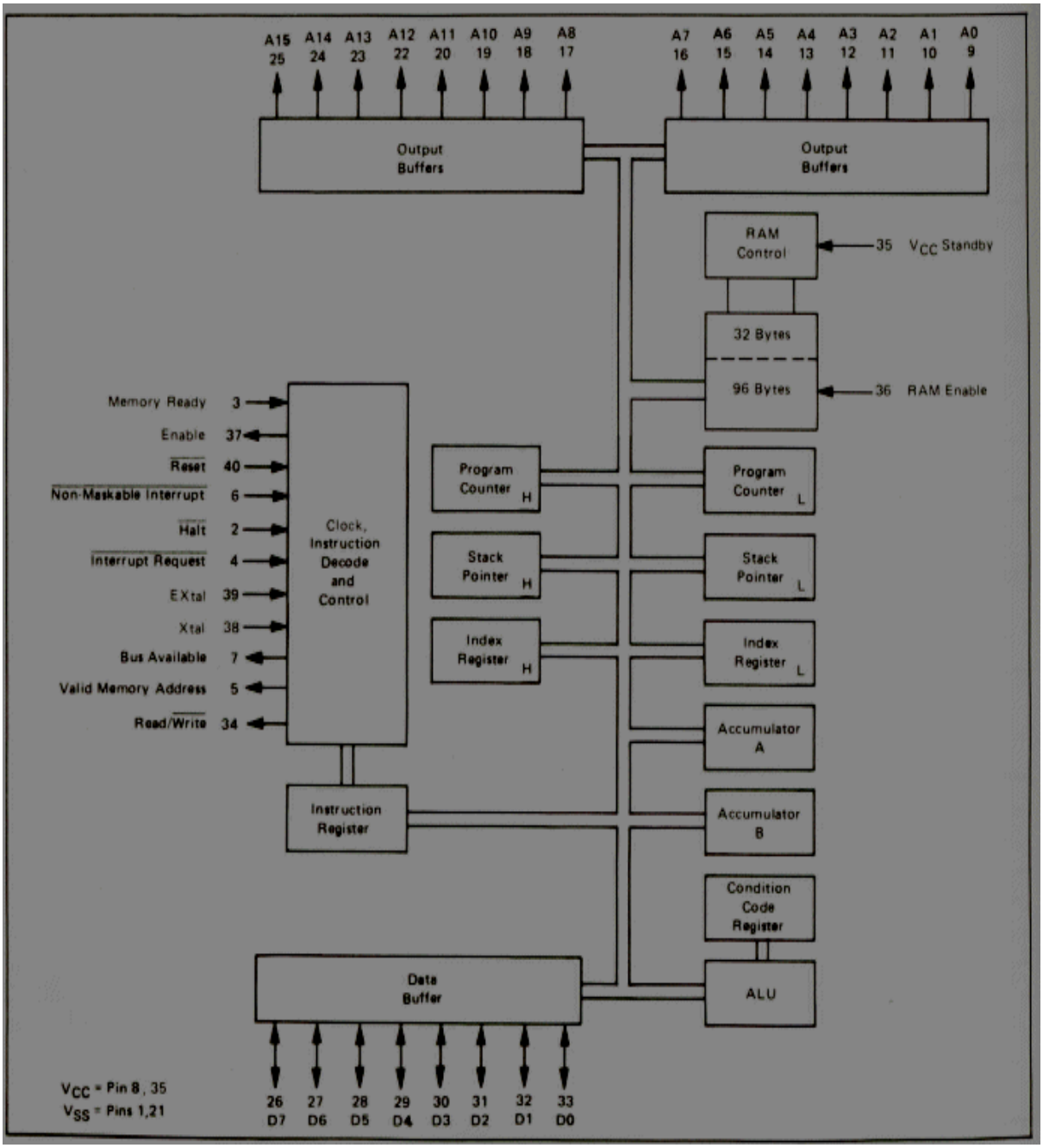
Thus -1 is represented (in the twos complement number system) by 11111111 = FF_{16}

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

SECTION 3

BASIC MICROPROCESSOR HARDWARE

Below is a block diagram of a 6800 8 bit microprocessor showing the address lines A0-A15, the internal registers, the control lines, the data lines and the interconnections of the internal elements.



Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

6800 Internal Registers

Central Processing Unit (CPU) REGISTERS

There are 6 registers available in the Motorola 6800 microprocessor: The Central Processing Unit (CPU) has three 16-bit registers and three 8-bit registers available for use by the programmer.

1. Two accumulators (ACCA and ACCB)

2. One index register (X)

3. One program counter register (PC)

4. One stack pointer register (SP).

5. One condition code register (CC)

Accumulators

The MPU contains 2 accumulators designated ACCA and ACCB. Each accumulator is 8 bits (one byte) long and is used to hold operands and data from the arithmetic logic unit.

Index Register

The index register (X) is a 16 bit (2 byte) register which is primarily used to store a memory address in the Indexed mode of memory addressing. The index register may be decremented, incremented and stored.

Program Counter

The program counter (PC) is a 16 bit register that contains the address of the next byte to be fetched from memory. When the current value of the program counter is placed on the address buss, the program counter will be incremented automatically.

Stack Pointer

The Stack Pointer (SP) is a 16 bit (2 byte) register that contains a beginning address, normally in RAM, where the status of the MPU registers may be stored when the MPU has other functions to perform, such as during an interrupt or during a Branch to Subroutine. The address in the stack pointer is the starting address of sequential memory locations in RAM where MPU status registers will be stored. In those applications that require storage of information in the stack when power is lost, the stack must be non-volatile.

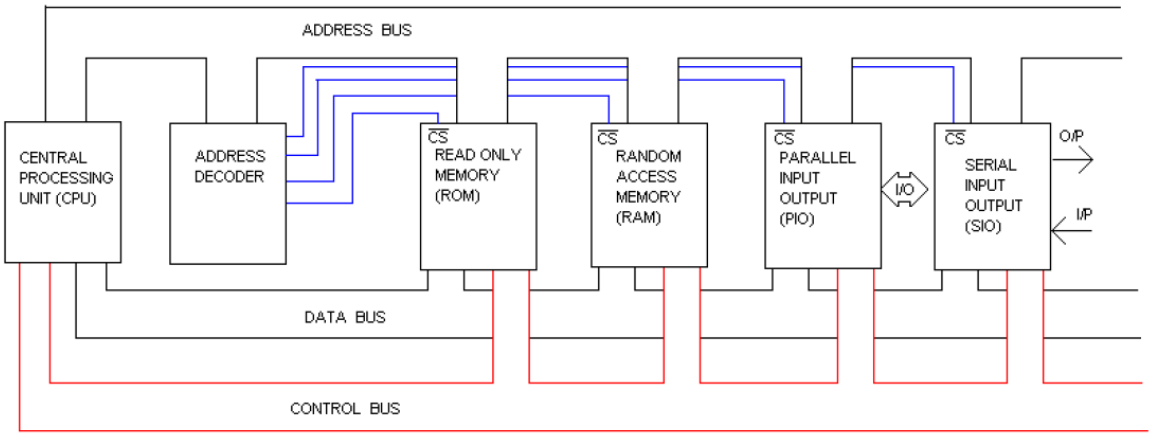
Condition Code Register (CC)

The condition code register is an 8 bit register. Each individual bit may get set or get cleared from execution of an instruction. Each instruction effects the condition code register differently. The primary use of this register is execution of the conditional branch instruction. Bit 6 and 7 are not used and remain at logic 1.

BIT (0-7) Number Function:

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

Computer Architecture



BLOCK DIAGRAM OF THE VON NEUMANN COMPUTER ARCHITECTURE

CPU - central processing unit, also MPU, microprocessing unit.

ADDRESS DECODER - used to select a particular address location. It sends a chip select (CS) signal to the block being addressed.

ROM - read only memory. In a small system the ROM would contain the monitor programme.

RAM - random access memory, used to store the user programme.

PIO - parallel input/output, is used to connect the system to external equipment, allowing data (and programs), to be moved in and out of the system.

SIO - serial input/output, is used to connect the system to external equipment via single pair connections.

BUS - a set of electrical connections, through which signals and power pass. The signals can be synchronous or asynchronous, usually the former, when they are controlled by a CLOCK signal.

DATA BUS - 8, 16, 32, 64 or more connections for bidirectional transmission of DATA. The larger the data bus, the more data can be transmitted.

ADDRESS BUS - 16, 32 or 64 connections which determine how much memory the computer CPU can 'address'. A 16 bit address bus can address 64 Kilobytes, a 32 bit bus 4 Gigabytes.

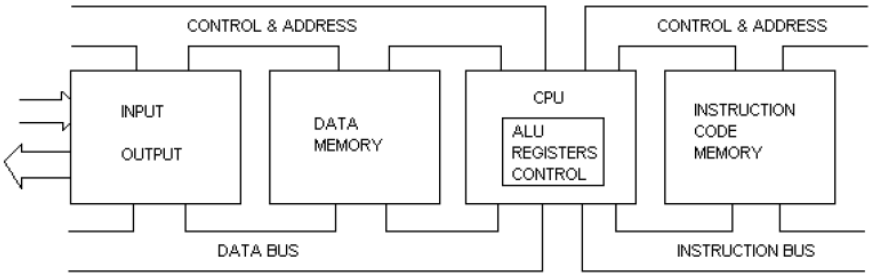
CONTROL BUS - all the other control signals and power for the cpu. Includes power, earth, clock signals, interrupts, controls for the other two buses and connections to other processors.

Comparison of Harvard and Von Neumann Architectures

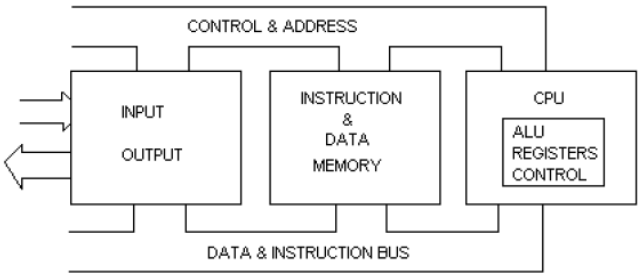
In Harvard architecture, the data bus and address bus are separate. Thus a greater flow of data is possible through the central processing unit, and of course, a greater speed of work. Separating a programme from data memory makes it further possible for instructions not to have to be 8-bit words. For example the Microchip PIC16F84 microcontroller uses 14 bits for instructions which allows for all instructions to be one word instructions. It is also typical for Harvard architecture to have fewer instructions than Von-Neumann's, and to have instructions usually executed in one cycle.

Microcontrollers with Harvard architecture are also called Reduced Instruction Set Computer (RISC) microcontrollers. Microprocessors with Von-Neumann's architecture are called Complex Instruction Set Computers (CISC).

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798



HARVARD ARCHITECTURE MICROPROCESSOR



VON NEUMANN ARCHITECTURE MICROPROCESSOR

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

SECTION 4

ADDRESSING MODES

Microprocessors have a number of different ways of fetching data from memory and placing data into registers or taking data from registers and placing into memory and these are called addressing modes. Most microprocessors have a number of different addressing modes and an instruction which for example places data into one of the accumulators can specify the location of this data either directly or indirectly.

The 6800 family addressing modes are as follows:

IMMEDIATE

DIRECT

EXTENDED

INDIRECT (or Indexed)

RELATIVE

IMPLIED

A brief description of each addressing mode as used in the 6800 family of 8 bit microprocessors follows:

IMMEDIATE

Using this addressing mode the data to be manipulated is supplied within the instruction. For example the instruction load accumulator A immediately, LDA #, is a two byte instruction and the second byte is the data to be placed in accumulator A. Thus the instruction LDA #$40 (the # symbol indicates the immediate addressing mode) will when executed during program execution result in the data $40 (01000000) being placed into the 8 bit accumulator A.

Similarly the three byte instruction LDX #$2057 will when executed during program execution result in the data $2057 (0010000001010111) being placed into the 16 bit index register X.

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

DIRECT AND EXTENDED

These addressing modes specify an address where the data to be manipulated is stored or is to be stored. The difference between them is that the extended mode specifies the full 16 bit address of the location and the direct mode only specifies the lower address byte. For the latter the upper address byte is retrieved from the DIRECT PAGE register and data can be placed in this register under program control.

As an example the instruction LDA $2057 (the $ symbol here indicates extended addressing) will when executed during program execution result in the data stored in location $2057 being placed into the 8 bit accumulator A.

The instruction LDA $49 (direct addressing and the $ symbol also indicates direct addressing) will when executed during program execution result in the data from a memory location (the location depends on the contents of the direct page register) being placed into the 8 bit accumulator A. If for example the direct page register contains $20 then the data from location $2049 will be copied into accumulator A.

INDEXED OR INDIRECT

In this addressing mode the address of the operand (data to be manipulated) is contained in a register, a 16 bit index register or in some microprocessors any of the 16 bit registers, and the indexed instruction specifies which register contains the data and also specifies an offset (usually an 8 bit value) which is added to the address to form the effective address (EA).

For example the instruction LDA 0,X (the X indicates indexed addressing) will when executed during program execution result in the data stored in the memory location pointed to by the X index register offset by 0 (ie no offset) being placed into the 8 bit accumulator A.

Similarly the instruction LDB $45,Y (the Y in this case specifies the Y index register) will when executed during program execution result in the data stored in the memory location pointed to by the Y index register offset by $45 being placed into the 8 bit accumulator B .

RELATIVE

The branching instructions use relative addressing where the destination address is an offset of the instruction location address. In the 6809 microprocessor the supplied offset can be up to 16 bits allowing branching to anywhere in memory. The original 6800 microprocessor only allowed an 8 bit twos complement offset restricting the branching range to +127 steps forward or -128 steps backward.

For example the instruction BRA $7F uses relative addressing to specify a branch to a location $7F ($127_{10}$) steps from the current location and the instruction BRA $80 uses relative addressing to specify a branch to a location $80 ($-128_{10}$) steps from the current location. (BRA stands for BRanch Always).

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

See relative addressing and conditional branching for more information.

IMPLIED

Here the action to be taken is implicit in the instruction and no further information needs to be supplied. For example the instruction CLRA (clear acc A) is a single byte implied addressing instruction which results in the accumulator A being reset (will now contain 00000000).

Similarly the single byte implied addressing instruction INCB (increment acc B) will when executed during program execution cause the B accumulator to be incremented (1 is added to it).

Other examples of implied addressing instructions are TBA (transfer contents of acc B to acc A) and DEX (decrement the index register X).

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

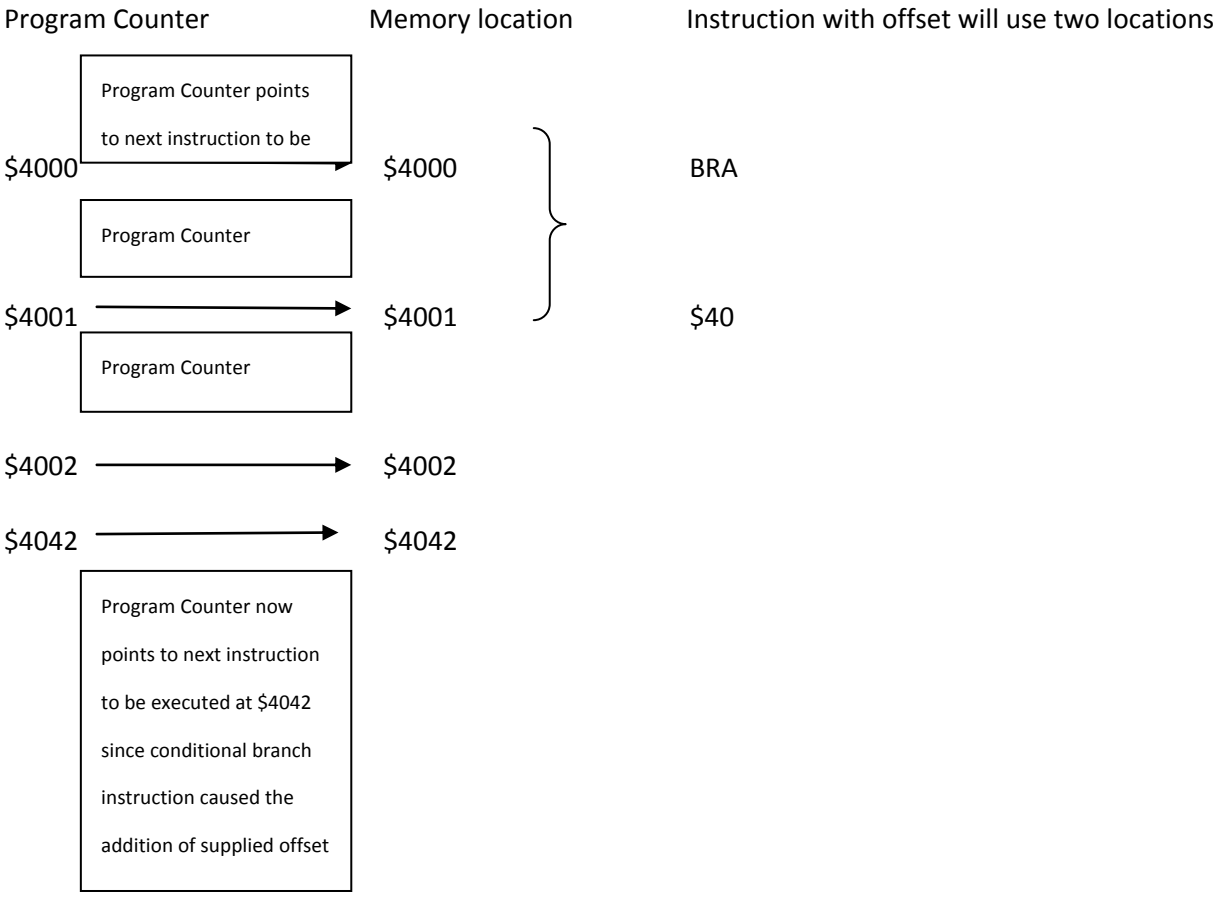
SECTION 5

Relative Addressing, Conditional Branching and the Condition Code Register

Relative addressing is the term applied to the addressing mode where the destination address is relative to the source location. In the 6800/6809 microprocessors this is done by supplying an OFFSET which is added to the contents of the program counter to determine the destination address. Since this offset is an 8 bit twos complement number representing decimal numbers in the range +127 to -128 branching can occur forwards or backwards within 255 locations of the location of the branching instruction. In hex notation the range is $7f to $80. Additionally in the 6809 there are long branch instructions which supply a four hex digit twos complement offset allowing branching to occur in the range +32768 to -32769 locations forwards/backwards. This gives a hex range of $7fff to $8000. For example if the microprocessor executes a branch always instruction (BRA) with a supplied offset of $40 and the branch instruction is located at $4000 then program execution will continue at

$4002 +$40 = $4042.

Note that the program counter will be pointing to $4002 when the microprocessor fetches the offset since the **program counter is always incremented after a 'fetch'.**



Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

Conditional branching is where the microprocessor must first check to see whether a condition has been met before branching occurs. The microprocessor uses the condition code register (CCR) to determine this. This is the decision making process in a computer – deciding whether to continue executing the current instructions or to branch to another location to execute different instructions.

For example if the microprocessor meets an instruction such as BEQ (branch if equal to zero) then the microprocessor examines the zero bit (or flag) in the CCR to determine if the result of the previous operation was zero and if so then the supplied offset is used to determine the address of the next instruction to be executed. If the result was not zero (CCR zero bit not set) then execution continues at the next location after the offset.

Condition code register (CCR)

The CCR is an internal microprocessor register which in the case of the 6800 microprocessor is an 8 bit register of which only 6 bits are used.

Structure of CCR in the 6800 microprocessor:

Not used	Not used	H	I	N	Z	V	C

Where H = half carry bit

I = interrupt bit

N= negative bit

Z + zero bit

V = twos complement overflow bit

C = carry bit

Arithmetic operations such as addition, decrementing, incrementing etc as well as loading and storing operations will set bits in the CCR according to the value of the result or number in question.

For example if the result of an arithmetic operation was zero then the Z bit in the CCR is set. Similarly if the result of an arithmetic operation is negative (MSB = 0) then the N bit is set. The carry bit is set if a carry occurs during addition and the same applies to the half carry bit (from the addition of the first two digits). The overflow bit indicates that the twos complement range has been exceeded such as when adding numbers which result in a number greater than +127. The interrupt bit is set by the user to indicate whether or not to allow an interrupt – if set an interrupt request is not allowed to interrupt the current operation.

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

For Example if the microprocessor adds the following hex numbers:

$7F + $01 = $80

But since $7F = $+127_{10}$ (the largest possible positive number using 8 bit twos complement notation) adding $01 to it results in a twos complement overflow hence the V bit in the CCR is set.

The microprocessor 'knows' this because the addition of two positive numbers (MSB = 0) should not result in a negative answer (MSB = 1).

Also since the result $80 is negative (= -128_{10}) the N bit in the CCR is set.

Also since the addition of the first two digits F and 1 results in a (half) carry the half carry bit is set.

CCR after this addition showing bits set (X = don't care).

		Half Carry		Negative		Overflow	
Not used	Not used	1	X	1	X	1	X

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

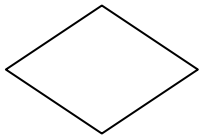
SECTION 6

FLOWCHART BASICS AND ASSEMBLY LANGUAGE PROGRAMS

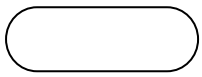
A flowchart is a visual representation of the flow of a program – it shows each step in the program and can also show decisions, conditional branches and jumps to subroutines.

In the flowcharts that follow the following conventions are used:

A rectangle is used to represent a process – either a single step such as fetching data from a memory location or a number of steps which perform a specific task.



This shape is used to represent a decision such as comparing a result with the desired result or checking to see if a result is zero and branching to a separate location based on this.

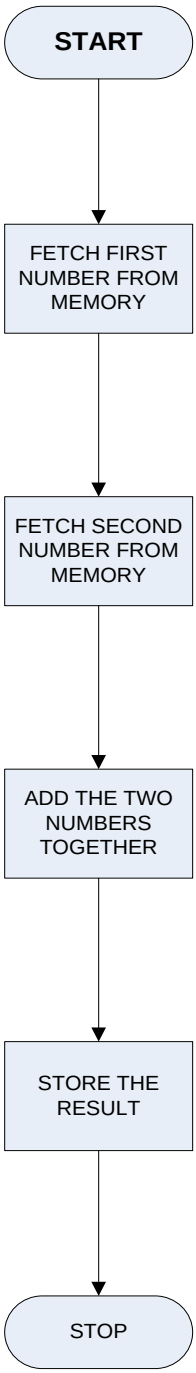


This shape represents either the start or the end of a program. The start can simply be a label but the end is always an instruction

There are many other flowchart symbols in use but these are the shapes used in the following examples.

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

A simple flowchart is shown below and in this procedure two numbers are fetched from two separate memory locations, added together and the result stored in a third memory location.



Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

ASSEMBLY LANGUAGE PROGRAM FOR ABOVE FLOWCHART

The actual memory locations to fetch the numbers from and store the result into are specified in the ***assembly language*** program. This is a sequence of assembly language instructions which perform the task.

All microprocessors have general purpose internal registers called **accumulators** into which data can be placed and on which arithmetic operations can be done (as well as numerous other operations).

In the program for the flowchart above data is fetched from the memory locations which we will call **First_Location** and **Second_Location** and placed into the two internal accumulators: ***accumulator A*** and ***accumulator B*** (these are often simply referred to as A and B).

This is done using **LOAD** instructions as shown below:

LDA $ First_Location

LDB $ Second_Location

The $ (hexadecimal) symbol above indicates the addressing mode used (in this case the extended addressing mode) and the label is the hexadecimal address of the memory location.

Note that the contents of the **First_Location** are copied into **accumulator A** using the **LDA** instruction and the contents of **Second_Location** are copied into **accumulator B** using the **LDB** instruction.

An addition operation is then done adding the contents of the two accumulators together. This is done using the **ABA** instruction (Add the contents of accumulator B to the contents of accumulator A or simply add B to A).

It is important to note that the result **accumulates in accumulator A.**

ABA

The result is then stored using the **STORE** instruction in this case the **STA** instruction since the result to be stored is in A (accumulator A).

As with the **LOAD** instructions above the **STORE** instruction has to specify where to store the result in this case in the **Result_Location**.

STA $ Result_Location

As above the $ symbol indicates an addressing mode – the extended addressing mode.

This copies the contents of **accumulator A** into the memory location specified **Result_Location.**

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

The final instruction is the **STOP** instruction. Depending on the system in use this can be either **SWI** (Software interrupt) or **RTS** (Return from Subroutine). In this case we will use the latter as in the system in use in the Labs all user programs are written as subroutines.

```
        RTS
```

The complete final program is shown below:

```
START:  LDA   $ First_Location     These instructions fetch the numbers to be added from
                                   their respective memory locations.
        LDB   $ Second_Location

        ABA                        This instruction adds the numbers.

        STA   $ Result_Location    This instruction stores the result in the specified memory
                                   location.

        RTS                        This causes the program to stop.
```

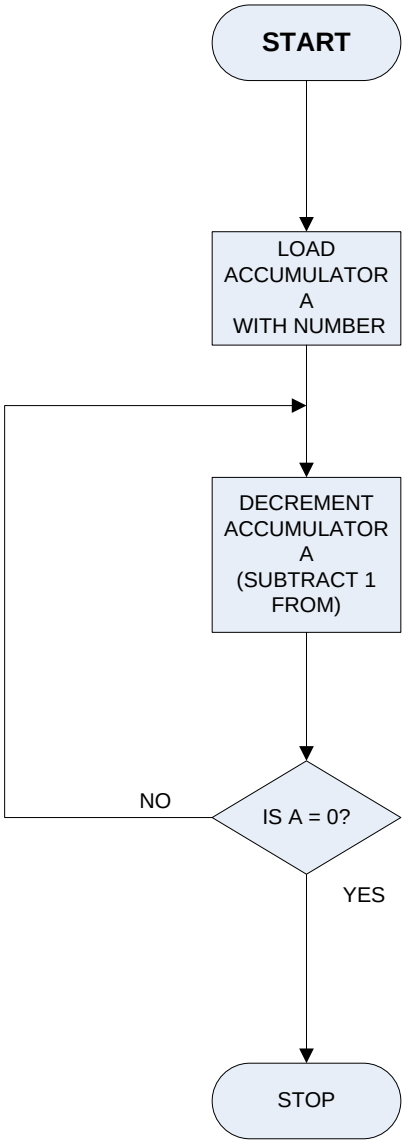
A FLOWCHART WITH DECISION (CONDITIONAL BRANCHING)

In this process an accumulator (accumulator A) is loaded with a number and then counted down to zero. Each time the accumulator is decremented (decrement = take 1 away) a check is made using a conditional branching instruction as to whether the result in accumulator A is zero or not. If the result is zero then the process stops and if the result is not zero the process repeats.

This is a simple **DELAY** process and the delay created depends on how long it takes the microprocessor to execute the instructions.

Microprocessors use delays for various reasons – for timing purposes in the case of output signals, for reading the state of switches which 'bounce', for displaying data etc.

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798



The program for the above flowchart is as follows:

START: LDA #$FF — This uses the **immediate addressing** mode to place $FF (11111111) into the 8 bit accumulator A. The # symbol indicates the immediate addressing mode.

LOOP: DECA — This subtracts 1 from accumulator A.

BNE LOOP — This is a **conditional branch** instruction which branches back to a place labelled LOOP if the result **is not zero.**

RTS — When the result is zero the process stops.

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

Note that this program also uses the LDA instruction but uses a different method of accessing the data – in this case the data is supplied with the instruction and is not fetched from a memory location (immediate addressing).

TO DETERMINE THE DELAY TIME

The delay created as mentioned depends on the time it takes the microprocessor to execute these instructions and this in turn depends on the speed of the microprocessor and the time taken to execute each instruction. Since $FF represents 255_{10} then the microprocessor executes the instructions in the loop 255 times and the total time taken to execute this program is:

Time taken to execute LDA instruction + (time taken for (DECA + BNE))*255 + time taken for RTS.

The time taken to execute individual instructions can be found in the **instruction set** for the microprocessor in use. For the 6800 microprocessor the instruction set can be found as an appendix to these notes and from this can be got the following information:

INSTRUCTION	NUMBER OF CLOCK CYCLES
LDA (IMMEDIATE)	2
DECA	2
BNE	2
RTS	5

The clock cycle time is the reciprocal of the clock frequency.

Knowing the clock frequency we can work out the total delay created by this program.

For example if the clock frequency is 1 MHz then the clock cycle time is 1/1 MHz = 1 µs.

Then knowing the clock cycle time we can work out the total delay time:

Total no of clock cycles = 2 + (2 +2)*255 + 5 = 1027 ⇨ delay is 1.027 ms.

To create a longer delay one of the 16 bit registers can be used for example the 16 bit index register (X) as demonstrated in the program below:

```
START:  LDX #$FFFF        This places the largest possible number into the X (index) register.
LOOP:   DEX               These instructions count down the X register to zero.
        BNE LOOP
        RTS
```

As an exercise determine the delay created by this program if the clock frequency is 20 MHz.

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

SECTION 7

MICROPROCESSOR INSTRUCTION SET AND PROGRAMMING THE MICROPROCESSOR

The instruction sets for the microprocessors referred to in these notes and used in the Laboratory are included below

From these instruction sets the following information can be found:

- A list of all available instructions for the microprocessor in question
- The shorthand assembly language notation for the instruction (called mnemonic)
- The addressing modes for each instruction
- The machine code for each instruction which is the actual data byte that the microprocessor fetches from memory, decodes and executes (called op code).
- The number of bytes in the instruction needed to determine the amount of memory that a program needs.
- The number of clock cycles that is required to execute each instruction
- Which bits in the Condition Code Register (CCR) are affected by each instruction
- Where applicable the action taken by each instruction.

As an example consider the instruction to load accumulator A which has the mnemonic LDAA (the mnemonic for the 6809 microprocessor is simplified to LDA).
From the 6800 instruction set the following information can be found for this instruction:
ADDRESSING MODES
There are four addressing modes available for this instruction and they are

- IMMEDIATE (the assembly language notation is LDAA #)
- DIRECT (the assembly language notation is LDAA $)
- EXTENDED (as above the assembly language notation is LDAA $)
- INDEXED (the assembly language notation is LDAA ?,X)

Hence there are four different ways of fetching data and placing it into accumulator A using this instruction and these are:

- The data can be supplied as part of the instruction (immediate) for example LDAA #$45 which tells the microprocessor to place $45 into acc A.
- The data can be in a memory location for example location $0045 and in this case either the direct (limited to the first 256 locations in memory in the case of the 6800 microprocessor) or the extended addressing mode can be used: LDAA $45 (direct) or LDAA $0045 (extended). Be aware

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

that LDAA #$45 ***is not the same as*** LDAA $45 as the first places supplied data into acc A and the second fetches data from a specified memory location and places it in acc A.

- The data can be in a memory location which is indirectly addressed by means of the indexed addressing mode. Here another internal register (the index register) contains the base address and the indexed addressing instruction LDAA ?, X tells the microprocessor to use the contents of the index register as the address from which to fetch data from (this can be offset by any value in the range $00 to $FF).

MACHINE CODES, NUMBER OF BYTES AND TIME TO EXECUTE

Each of the instructions above has a different machine code and size. From the instruction set can be found the following information:

INSTRUCTION	MACHINE CODE	SIZE IN BYTES	TIME TAKEN TO EXECUTE
LDAA # (IMMEDIATE)	86	2	2 (CLOCK CYCLES)
LDAA $ (DIRECT)	96	2	3
LDAA $ (EXTENDED)	B6	4	3
LDAA ?,X (INDEXED)	A6	2	5

ACTION TAKEN

The action taken in all cases is that data from a memory location is copied into acc A.

CONDITION CODE REGISTER

The action of placing data into acc A will affect the condition code register (CCR) The data in question is examined and if it is negative (MSB =1) then the N bit is set and if zero then the Z bit is set. The overflow bit is reset and the other bits are not affected.

MACHINE CODE

Most programs are written in assembly language using the mnemonics referred to above. A computer program which converts this assembly language program into the relevant machine code for the microprocessor is called an assembler. Once the program is in machine code it can be placed into (or downloaded to) the microprocessors memory. The microprocessor can then run the program from memory.

Older systems required that the machine code be entered manually into the microprocessors memory one byte at a time and thus it was necessary to, using the instruction set, convert the program into machine code.

As an example of converting a program to machine code consider the simple program to load and add two numbers as covered in section 6 but in this case adding together the contents of the two memory locations $4000 and $4001 and storing the result in memory location $4002.

This program must be placed in memory and assume that the start address for the program is to be $6789.

From the instruction set for the 6800 microprocessor the following machine codes can be found for the assembly language instructions:

INSTRUCTION	OP CODE	OPERAND
LDA $4000 (extended addressing)	B6	4000
LDB $4001 (extended addressing)	F6	4001
ABA (implied addressing)	1B	
STA $4002 (extended addressing)	B7	4002
RTS (implied addressing)	39	

The op code is the machine code for the instruction and the operand is data supplied with it to be operated on – either a memory location or a data byte.

Since the program is to be located starting at memory location $6789 then the following data must be placed into the following memory locations:

MEMORY LOCATION	DATA
$6789	B6
$678A	40
$678B	00
$678C	F6
$678D	40
$678E	01
$678F	1B
$6790	B7
$6791	40
$6792	02
$6793	39

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

Often this is presented in an easier to read/understand form as shown below
WHERE THE START MEMORY LOCATION FOR EACH INSTRUCTION ONLY IS SHOWN:

MEMORY LOCATION	DATA
$6789	86 40 00
$678C	B6 40 01
$678F	1B
$6790	B7 40 02
$6793	39

Note the hexadecimal numbering system for the memory locations.
Note also that it is not known what numbers are added together or what the result will be as the program adds together the ***contents*** of the memory locations.

As an example of converting to machine code consider the simple delay program covered in section 6:

DELAY PROGRAM

```
START: LDA #$FF
LOOP:  DECA
       BNE LOOP
       RTS
```

This uses the **immediate addressing** mode to place $FF (11111111) into the 8 bit accumulator A. The # symbol indicates the immediate addressing mode.

This subtracts 1 from accumulator A.

This is a **conditional branch** instruction which branches back to a place labelled LOOP if the result **is not zero.**

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

MACHINE CODE

Assuming that the program is to be located starting at $A691 and using the instruction set the following machine codes will be placed in the following locations:

INSTRUCTION	LOCATION	MACHINE CODE
LDAA #$FF	$A691	86 FF
LOOP DECA	**$A693**	**4A**
BNE (LOOP)	**$A694**	**26 FD**
RTS	**$A696**	**39**

The conditional branching instruction BNE (branch if not equal to zero) branches back 3 steps to the DECA instruction if the condition is not true (here if the result in Acc A is *not zero* branching will occur). This offset is supplied as a two's complement negative number (– 3 = $FD) and if branching is to occur then this is added to the current contents of the program counter ($A696) to cause program execution to continue at $A693

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

6800 Instruction Set

Operation	Mnem.	Immed.	Direct	Index	Extend	Inher.	Operation	CC Reg
		OP·~·#	OP·~·#	OP·~·#	OP·~·#	OP·~·#		HINZVC
Add	ADDA	8B·2·2	9B·3·2	AB·5·2	BB·4·3	· ·	A=A+M	T•TTTT
	ADDB	CB·2·2	DB·3·2	EB·5·2	FB·4·3	· ·	B=B+M	T•TTTT
Add Accumulators	ABA	· ·	· ·	· ·	· ·	1B·2·1	A=A+B	T•TTTT
Add with Carry	ADCA	89·2·2	99·3·2	A9·5·2	B9·4·3	· ·	A=A+M+C	T•TTTT
	ADCB	C9·2·2	D9·3·2	E9·5·2	F9·4·3	· ·	B=B+M+C	T•TTTT
And	ANDA	84·2·2	94·3·2	A4·5·2	B4·4·3	· ·	A=A.M	••TTR•
	ANDB	C4·2·2	D4·3·2	E4·5·2	F4·4·3	· ·	B=B.M	••TTR•
Bit Test	BITA	85·2·2	95·3·2	A5·5·2	B5·4·3	· ·	A.M	••TTR•
	BITB	C5·2·2	D5·3·2	E5·5·2	F5·4·3	· ·	B.M	••TTR•
Clear	CLR	· ·	· ·	6F·7·2	7F·6·3	· ·	M=00	••RSRR
	CLRA	· ·	· ·	· ·	· ·	4F·2·1	A=00	••RSRR
	CLRB	· ·	· ·	· ·	· ·	5F·2·1	B=00	••RSRR
Compare	CMPA	81·2·2	91·3·2	A1·5·2	B1·4·3	· ·	A-M	••TTTT
	CMPB	C1·2·2	D1·3·2	E1·5·2	F1·4·3	· ·	B-M	••TTTT
Compare Accumulators	CBA	· ·	· ·	· ·	· ·	11·2·1	A-B	••TTTT
Complement 1's	COM	· ·	· ·	63·7·2	73·6·3	· ·	M=-M	••TTRS
	COMA	· ·	· ·	· ·	· ·	43·2·1	A=-A	••TTRS
	COMB	· ·	· ·	· ·	· ·	53·2·1	B=-B	••TTRS
Complement 2's	NEG	· ·	· ·	60·7·2	70·6·3	· ·	M=00-M	••TT12
	NEGA	· ·	· ·	· ·	· ·	40·2·1	A=00-A	••TT12
	NEGB	· ·	· ·	· ·	· ·	50·2·1	B=00-B	••TT12
Decimal Adjust	DAA	· ·	· ·	· ·	· ·	19·2·1	*	••TTT3
Decrement	DEC	· ·	· ·	6A·7·2	7A·6·3	· ·	M=M-1	••TT4•
	DECA	· ·	· ·	· ·	· ·	4A·2·1	A=A-1	••TT4•
	DECB	· ·	· ·	· ·	· ·	5A·2·1	B=B-1	••TT4•
Exclusive OR	EORA	88·2·2	98·3·2	A8·5·2	B8·4·3	· ·	A=A(+)M	••TTR•
	EORB	C8·2·2	D8·3·2	E8·5·2	F8·4·3	· ·	B=B(+)M	••TTR•
Increment	INC	· ·	· ·	6C·7·2	7C·6·3	· ·	M=M+1	••TT5•
	INCA	· ·	· ·	· ·	· ·	4C·2·1	A=A+1	••TT5•
	INCB	· ·	· ·	· ·	· ·	5C·2·1	B=B+1	••TT5•
Load Accumulator	LDAA	86·2·2	96·3·2	A6·5·2	B6·4·3	· ·	A=M	••TTR•
	LDAB	C6·2·2	D6·3·2	E6·5·2	F6·4·3	· ·	B=M	••TTR•
OR, Inclusive	ORAA	8A·2·2	9A·3·2	AA·5·2	BA·4·3	· ·	A=A+M	••TTR•
	ORAB	CA·2·2	DA·3·2	EA·5·2	FA·4·3	· ·	B=B+M	••TTR•
Push Data	PSHA	· ·	· ·	· ·	· ·	36·4·1	Msp=A, *-	••••••
	PSHB	· ·	· ·	· ·	· ·	37·4·1	Msp=B, *-	••••••
Pull Data	PULA	· ·	· ·	· ·	· ·	32·4·1	A=Msp, *+	••••••
	PULB	· ·	· ·	· ·	· ·	33·4·1	B=Msp, *+	••••••
Rotate Left	ROL	· ·	· ·	69·7·2	79·6·3	· ·	Memory *1	••TT6T
	ROLA	· ·	· ·	· ·	· ·	49·2·1	Accum A *1	••TT6T
	ROLB	· ·	· ·	· ·	· ·	59·2·1	Accum B *1	••TT6T
Rotate Right	ROR	· ·	· ·	66·7·2	76·6·3	· ·	Memory *2	••TT6T
	RORA	· ·	· ·	· ·	· ·	46·2·1	Accum A *2	••TT6T
	RORB	· ·	· ·	· ·	· ·	56·2·1	Accum B *2	••TT6T
Arithmetic Shift Left	ASL	· ·	· ·	68·7·2	78·6·3	· ·	Memory *3	••TT6T
	ASLA	· ·	· ·	· ·	· ·	48·2·1	Accum A *3	••TT6T
	ASLB	· ·	· ·	· ·	· ·	58·2·1	Accum B *3	••TT6T
Arithmetic Shift Right	ASR	· ·	· ·	67·7·2	77·6·3	· ·	Memory *4	••TT6T
	ASRA	· ·	· ·	· ·	· ·	47·2·1	Accum A *4	••TT6T
	ASRB	· ·	· ·	· ·	· ·	57·2·1	Accum B *4	••TT6T
Logic Shift Right	LSR	· ·	· ·	64·7·2	74·6·3	· ·	Memory *5	••TT6T
	LSRA	· ·	· ·	· ·	· ·	44·2·1	Accum A *5	••TT6T
	LSRB	· ·	· ·	· ·	· ·	54·2·1	Accum B *5	••TT6T
Store Accumulator	STAA	· ·	97·4·2	A7·6·2	B7·5·3	· ·	M=A	••TTR•
	STAB	· ·	D7·4·2	E7·6·2	F7·5·3	· ·	M=B	••TTR•
Subtract	SUBA	80·2·2	90·3·2	A0·5·2	B0·4·3	· ·	A=A-M	••TTTT

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

	SUBB	C0·2·2	D0·3·2	E0·5·2	F0·4·3	· ·	B=B-M	••TTTT
Subtract Accumulators	SBA	· ·	· ·	· ·	· ·	10·2·1	A=A-B	••TTTT
Subtract with Carry	SBCA	82·2·2	92·3·2	A2·5·2	B2·4·3	· ·	A=A-M-C	••TTTT
	SBCB	C2·2·2	D2·3·2	E2·5·2	F2·4·3	· ·	B=B-M-C	••TTTT
Transfer Accumulators	TAB	· ·	· ·	· ·	· ·	16·2·1	B=A	••TTR•
	TBA	· ·	· ·	· ·	· ·	17·2·1	A=B	••TTR•
Test, Zero/Minus	TST	· ·	· ·	6D·7·2	7D·6·3	· ·	M-00	••TTRR
	TSTA	· ·	· ·	· ·	· ·	4D·2·1	A-00	••TTRR
	TSTB	· ·	· ·	· ·	· ·	5D·2·1	B-00	••TTRR

Operation	Mnem.	Immed.	Direct	Index	Extend	Inher.	Operation	CC Reg
Compare Index Register	CPX	8C·3·3	9C·4·2	AC·6·2	BC·5·3	· ·	Formula 1	••7T8•
Decrement Index Register	DEX	· ·	· ·	· ·	· ·	09·4·1	X=X-1	•••T••
Dec Stack Pointer	DES	· ·	· ·	· ·	· ·	34·4·1	SP=SP-1	••••••
Inc Index Regster	INX	· ·	· ·	· ·	· ·	08·4·1	X=X+1	•••T••
Inc Stack Pointer	INS	· ·	· ·	· ·	· ·	31·4·1	SP=SP+1	••••••
Load Index Register	LDX	CE·3·3	DE·4·2	EE·6·2	FE·5·3	· ·	Formula 2	••9TR•
Load Stack Pointer	LDS	8E·3·3	9E·4·2	AE·6·2	BE·5·3	· ·	Formula 3	••9TR•
Store Index Register	STX	· ·	DF·5·2	EF·7·2	FF·6·3	· ·	Formula 4	••9TR•
Store Stack Pointer	STS	· ·	9F·5·2	AF·7·2	BF·6·3	· ·	Formula 5	••9TR•
Index Reg > Stack Pnter	TXS	· ·	· ·	· ·	· ·	35·4·1	SP=X-1	••••••
Stack Ptr > Index Regtr	TSX	· ·	· ·	· ·	· ·	30·4·1	X=SP+1	••••••

Operation: Branch If	Mnem.	Immed.	Direct	Index	Extend	Inher.	Operation	CC Reg
Always	BRA	· ·	20·4·2	· ·	· ·	· ·	none	••••••
Carry is Clear	BCC	· ·	24·4·2	· ·	· ·	· ·	C=0	••••••
Carry is Set	BCS	· ·	25·4·2	· ·	· ·	· ·	C=1	••••••
Equals Zero	BEQ	· ·	27·4·2	· ·	· ·	· ·	Z=1	••••••
Greater or Equal to Zero	BGE	· ·	2C·4·2	· ·	· ·	· ·	N(+)V=0	••••••
Greater than Zero	BGT	· ·	2E·4·2	· ·	· ·	· ·	Z+N(+)V=0	••••••
Higher	BHI	· ·	22·4·2	· ·	· ·	· ·	C+Z=0	••••••
Less or Equal than Zero	BLE	· ·	2F·4·2	· ·	· ·	· ·	Z+N(+)V=1	••••••
Lower or Same	BLS	· ·	23·4·2	· ·	· ·	· ·	C+Z=1	••••••
Less Than Zero	BLT	· ·	2D·4·2	· ·	· ·	· ·	N(+)V=1	••••••
Minus	BMI	· ·	2B·4·2	· ·	· ·	· ·	N=1	••••••
Not Zero	BNE	· ·	26·4·2	· ·	· ·	· ·	Z=0	••••••
Overflow Clear	BVC	· ·	28·4·2	· ·	· ·	· ·	V=0	••••••
Overflow Set	BVS	· ·	29·4·2	· ·	· ·	· ·	V=1	••••••
Plus	BPL	· ·	2A·4·2	· ·	· ·	· ·	N=0	••••••

Operation	Mnem.	Immed.	Direct	Index	Extend	Inher.	Operation	CC Reg
Branch to Subroutine	BSR	· ·	8D·8·2	· ·	· ·	· ·		••••••
Jump	JMP	· ·	· ·	6E·4·2	7E·3·3	· ·		••••••
Jump to Subroutine	JSR	· ·	· ·	AD·8·2	BD·9·3	· ·		••••••
No Operation	NOP	· ·	· ·	· ·	· ·	01·2·1		••••••
Return from Interrupt	RTI	· ·	· ·	· ·	· ·	3B·A·1		AAAAAA
Return from Subroutine	RTS	· ·	· ·	· ·	· ·	39·5·1		••••••
Software Interrupt	SWI	· ·	· ·	· ·	· ·	3F·C·1		•S••••
Wait For Interrupt	WAI	· ·	· ·	· ·	· ·	3E·9·1		•B••••

Operation	Mnem.	Immed.	Direct	Index	Extend	Inher.	Operation	CC Reg

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

Clear Carry	CLC	·	·	·	·	·	·	0C·2·1	C=0	•••••R
Clear Interrupt	CLI	·	·	·	·	·	·	0E·2·1	I=0	•R••••
Clear Overflow	CLV	·	·	·	·	·	·	0A·2·1	V=0	••••R•
Set Carry	SEC	·	·	·	·	·	·	0D·2·1	C=1	•••••S
Set Interrupt	SEI	·	·	·	·	·	·	0F·2·1	I=1	•S••••
Set Overflow	SEV	·	·	·	·	·	·	0B·2·1	V=1	••••S•
CCR=Accumulator A	TAP	·	·	·	·	·	·	06·2·1	CCR=A	CCCCCC
Accumlator A=CCR	TPA	·	·	·	·	·	·	07·2·1	A=CCR	••••••

OP Operation Code, in Hexadecimal
~ Number of MPU cycles required
# Number of program bytes required
+ Arithmetic Plus
- Arithmetic Minus
+ Boolean AND
Msp Contents of Memory pointed to be Stack Pointer
+ Boolean Inclusive OR
(+) Boolean Exclusive OR (XOR)
* Converts Binary Addition of BCD Characters into BCD Format
*- SP=SP-1
*+ SP=SP+1

Condition Code Register Legend
• Not Affected
R Reset (0, Low)
S Set (1, High)
T Tests and sets if True, cleared otherise
1 Test: Result=10000000?
2 Test: Result=00000000?
3 Test: Decimal value of most significant BCD character greater than nine?
(Not cleared if previously set)
4 Test: Operand=10000000 prior to execution?
5 Test: Operand=01111111 prior to execution?
6 Test: Set equal to result or N(+)C after shift has occurred.
7 Test: Sign bit of most significant byte or result=1?
8 Test: 2's compliment overflow from subtraction of least
significant bytes?
9 Test: Result less than zero? (Bit 15=1)
A Load Condition Code Register from Stack.
B Set when interrupt occurs. If previously set, a NMI is
required to exit the wait state.
C Set according to the contents of Accumulator A.

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

SECTION 8

ADDRESS DECODING

Microcomputer systems use a variety of different memory elements such as RAM for the temporary storage of data during program execution, ROM for the permanent storage of programs, EPROM for development purposes and in some systems memory mapped I/O devices.

A system is thus required to select one of a number of memory devices based of the address of that device and this is the purpose of address decoding hardware. In general the upper address bits are used via decoders to select or enable individual devices and then the lower address bits are used to select or address memory locations within that device.

The number of address bits (or lines) needed to select a location within a device depends on the size of the device (memory capacity) and this in turn determines the number of address bits remaining to be used for decoding purposes.

The table below shows the number of address lines required for common memory capacities used in simple microcomputer systems.

MEMORY CAPACITY

64k	32k	16k	8k	4k	2k	1k	512	256	128	64	32	16	8	4	2
16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

NUMBER OF ADDRESS LINES

Note that for 16k for example means that there are 16384 locations and 14 address lines are needed to select any one of these locations. (2^{14} = 16384).

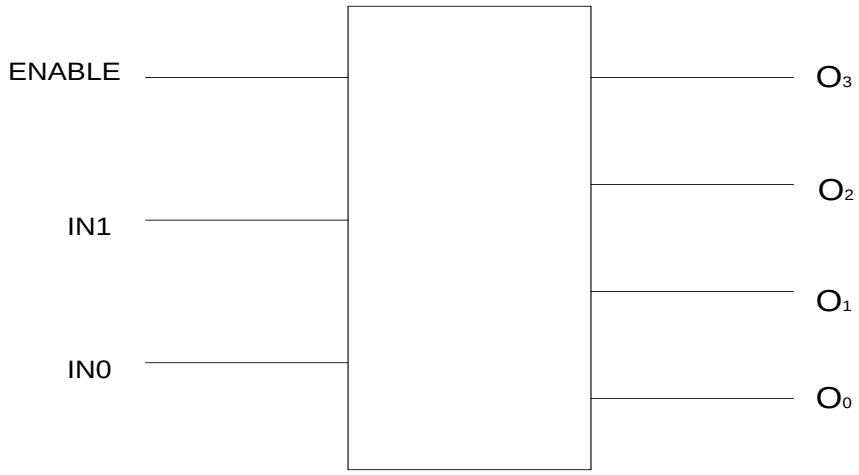
With 16 address lines the maximum number of locations is 2^{16} = 65536 which is referred to as 64k (k stands for kilo which is 1024 in digital notation). Also note that for 8 bit microcomputer systems each location can store one byte (8 bits) and so the notation 64 kbyte is often used.

For simple 8 bit, 16 address line microprocessors 2 to 4 line, 3 to 8 line or 4 to 16 line decoders can be used. These decoders use address lines as select inputs to select or enable one of the output lines. The logic symbol and truth table for a 2 to 4 line decoder is shown below.

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

This decoder is enabled by a low on its ENABLE input and when enabled one of its output lines will be high (or low) depending on the logic levels on the select lines (IN1 and IN0).

Logic Symbol for 2 to 4 line decoder:



Once the device is enabled the two select lines IN0 and IN1 determine which of the four output lines is set high according to the table below.

Truth Table for 2 to 4 line decoder with active low enable input

INPUTS			OUTPUTS			
ENABLE	IN1	IN0	O_3	O_2	O_1	O_0
0	0	0	0	0	0	1
0	0	1	0	0	1	0
0	1	0	0	1	0	0
0	1	1	1	0	0	0
1	0	0	DIS	DIS	DIS	DIS
1	0	1	DIS	DIS	DIS	DIS
1	1	0	DIS	DIS	DIS	DIS
1	1	1	DIS	DIS	DIS	DIS

DIS = Disabled with output in Tri State mode

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

ADDRESS DECODING EXAMPLE

This shows by means of a block diagram of the address decoding system how two of the above decoders could be used to implement the memory map as shown in Figure 1 below.

MEMORY RANGE	MEMORY TYPE
0000 – 1FFF	RAM1
2000 – 2FFF	RAM2
4800 – 48FF	ROM1
5800 – 58FF	ROM2

Figure 1

Memory Map for System

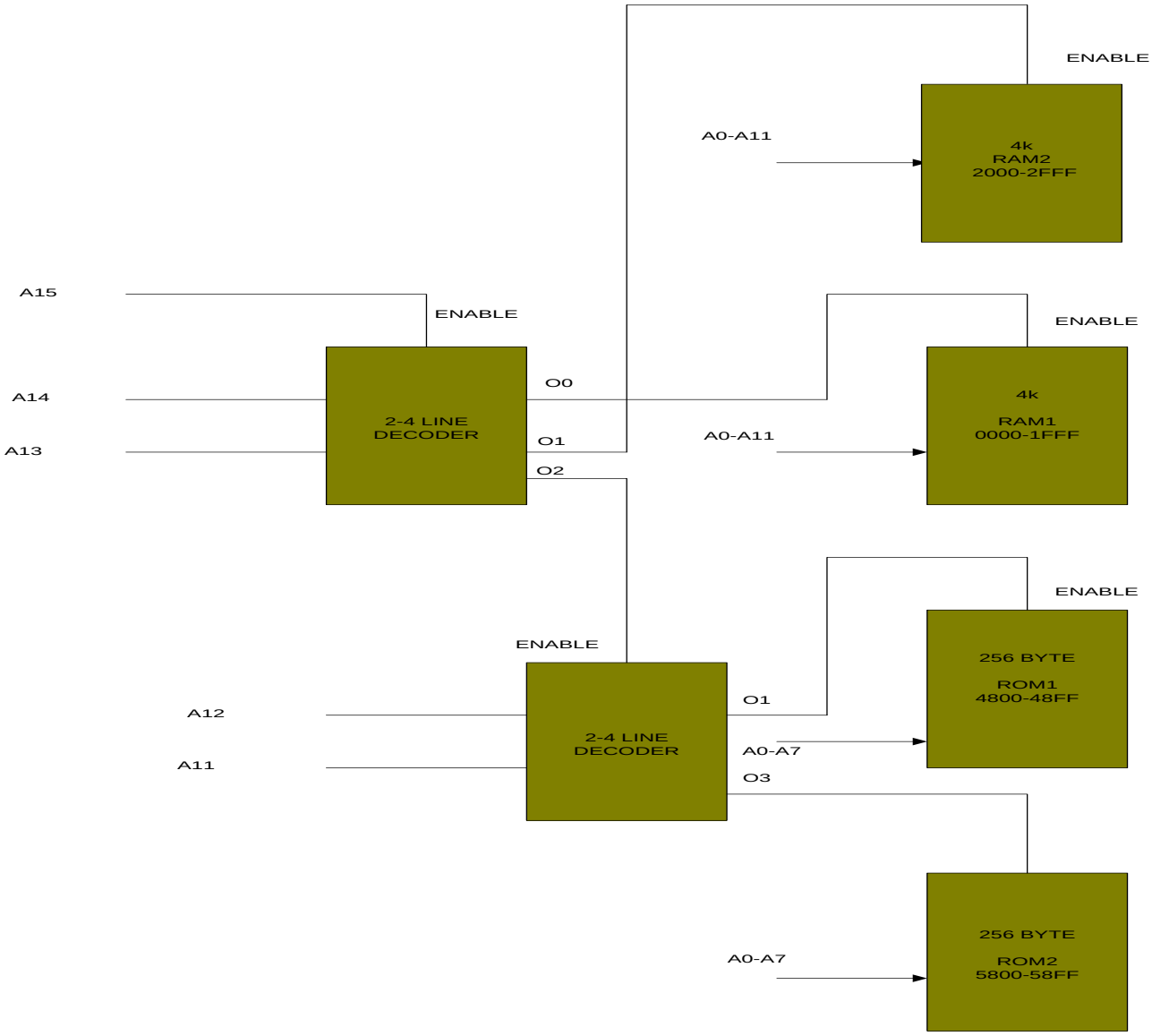
RANGE	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1FFF	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
2000	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2FFF	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1
4800	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
48FF	0	1	0	0	1	0	0	0	1	1	1	1	1	1	1	1
5800	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0
58FF	0	1	0	1	1	0	0	0	1	1	1	1	1	1	1	1

From the above table it can be seen that A_{15} can be used as the enable input, A_{14} and A_{13} can be used to select one of the two decoders and A_{12} and A_{11} can be used to select one of the two ROM's as shown in the block diagram below.

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

Note that in the case of ROM selection three of the address bits are not used – A8, A9 and A10. This means that these bits can be at any logic level (0 or 1) and it will have no effect on the address decoding. Hence any address in the range $48XX to $4FXX will select ROM1 and similarly any address in the range $58XX to $5FXX will select ROM2. This is called **REDUNDANT ADDRESSING** and obviously is undesirable since these redundant addresses cannot be used to select other memory elements.

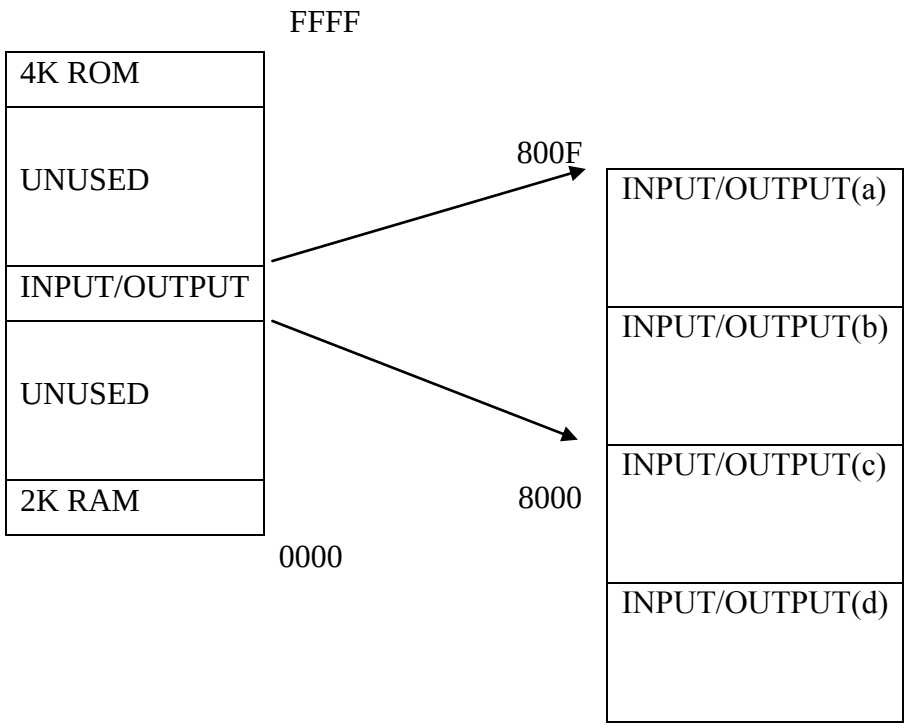
X = don't care ie can be any hex value.



Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

ADDRESS DECODING EXAMPLE2

The following shows the address decoding implementation for the memory map shown below.



Memory map in binary form shown below in Table (1)

RANGE	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
07FF	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
8000	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
800F	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
F000	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
FFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table (1)

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

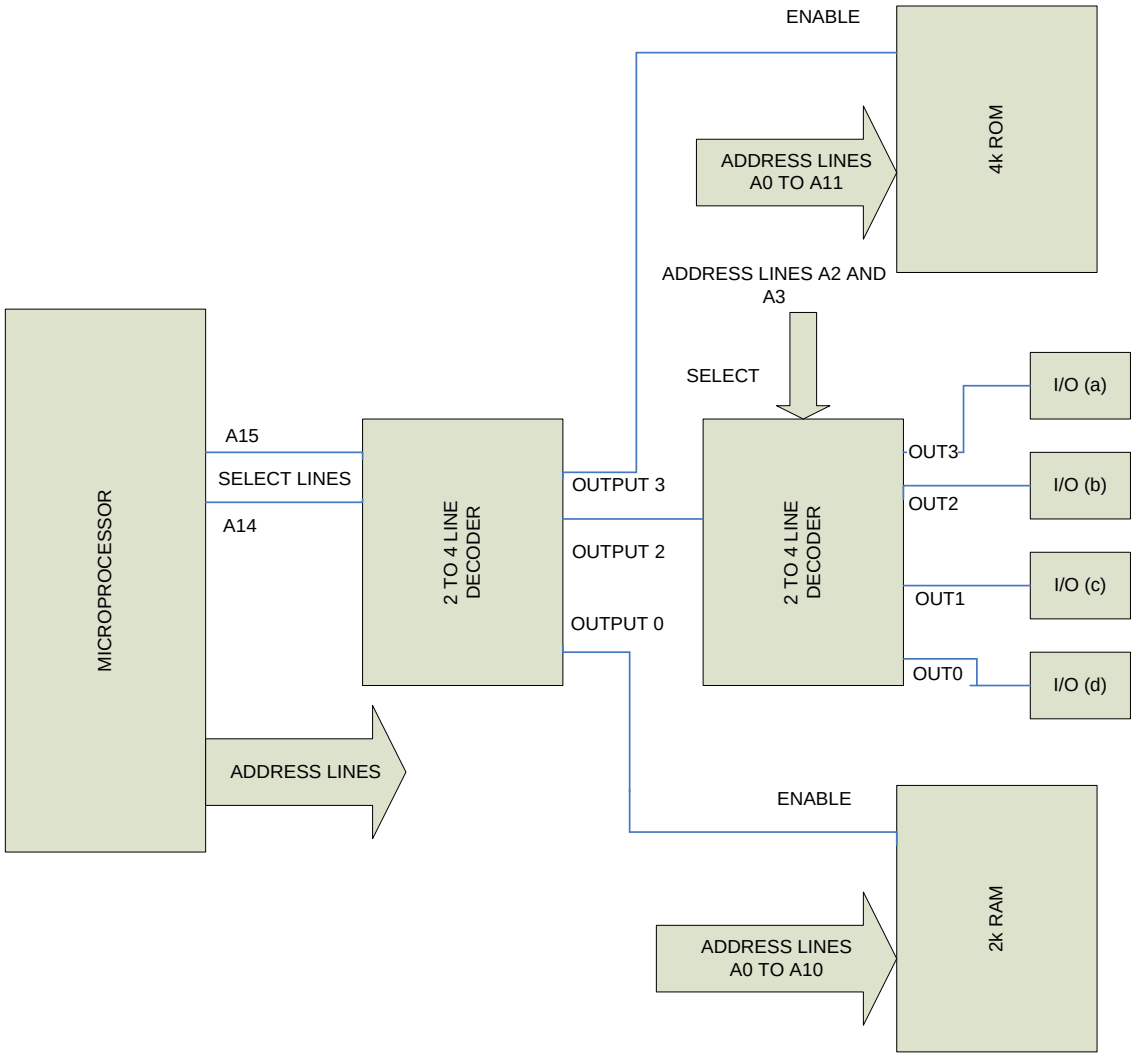
From the above table it can be seen that A_{15} and A_{14} can be used to select one of three 'elements', 2k RAM, I/O or 4K ROM. From the I/O table below (Table (2)) it can be seen that address lines A2 and A3 can be used to select one of four I/O ports each allocated or mapped to the addresses shown.

I/O ADDRESS TABLE

HEX	A15 – A4	A3	A2	A1	A0
8000	⟷	0	0	0	0
8003	⟷	0	0	1	1
8004	⟷	0	1	0	0
8007	⟷	0	1	1	1
8008	⟷	1	0	0	0
800B	⟷	1	0	1	1
800C	⟷	1	1	0	0
800F	⟷	1	1	1	1

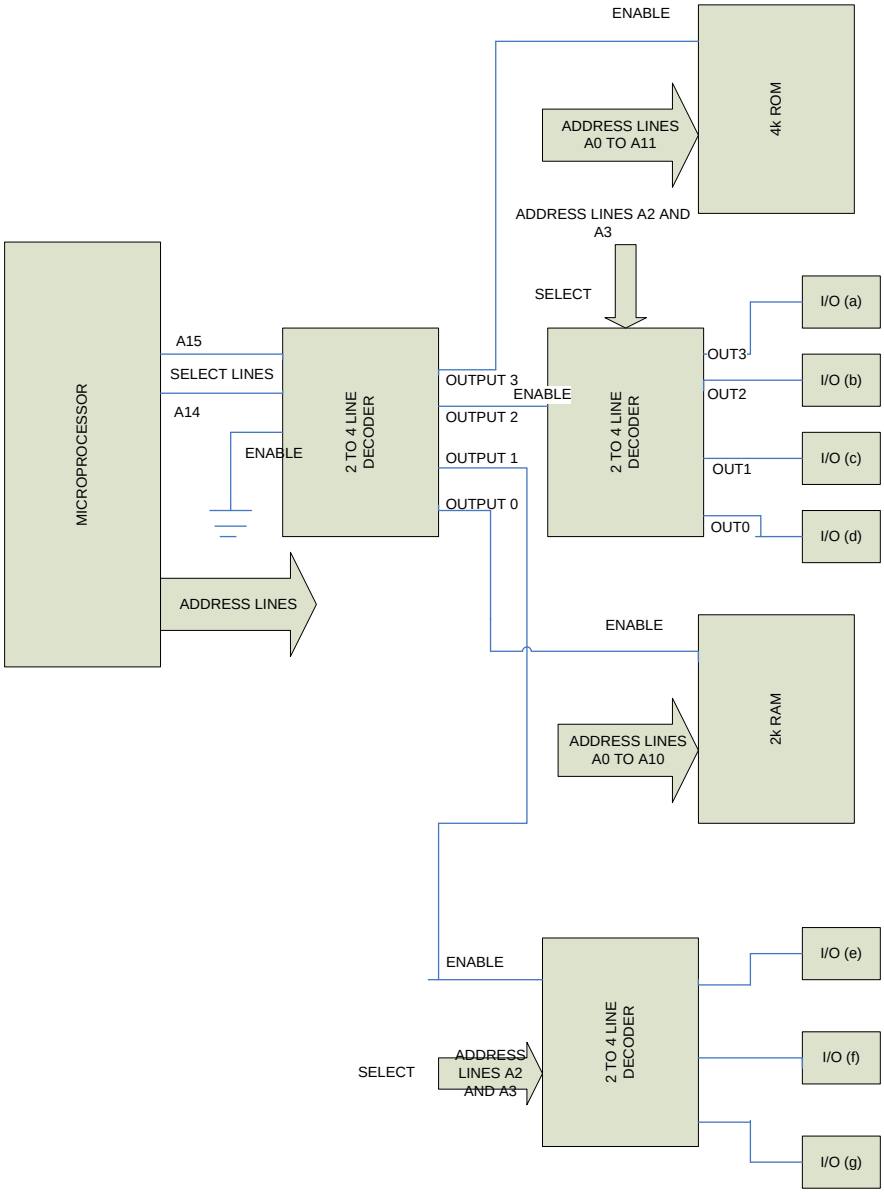
Table (2)

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798



Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

Adding three more four port I/O devices



Address ranges for new devices

DEVICE NAME	ADDRESS RANGE
I/O (e)	4008 TO 400B
I/O (f)	4004 TO 4007
I/O (g)	4000 TO 4003

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

SECTION 9
MEMORY AND MEMORY TESTING

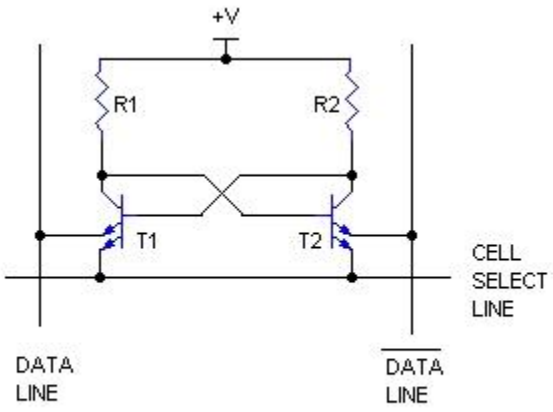
Random Access Memory (RAM)

SRAM (Static RAM):

SRAM is a type of RAM that holds data, placed within its cells, until the values are either overwritten or the power is removed. This is opposed to Dynamic RAM (DRAM), which allows the data to exit by the cells discharging, every few milliseconds unless it is refreshed.

In static RAM, a form of flip-flop holds each bit of memory. A flip-flop for a memory celllusuallyt takes four or six transistors, but never has to be refreshed. This makes static RAM significantly faster, with access times in the 10 to 30-nanosecond range, than DRAM. However, because it has more parts, a static memory cell takes up more space on a chip than a dynamic memory cell. Therefore, you get less memory per chip, which makes static RAM more expensive.

The circuit schematic shown below utilises bipolar dual emitter devices:



STATIC RAM BIPOLAR TRANSISTORS

In STANDBY MODE the cell select line is held at a lower potential (0.3 Volts) than the DATA or /DATA lines (1 to 2 Volts). So current goes through the cell select line from the ON transistor and DATA and /DATA lines are in high impedance state.

To READ data the cell select line is raised high (3 Volts) and DATA and /DATA lines are fed to a differential amplifier which gives a high or low out, depending on the state of T1 and T2.

To WRITE data the cell select line is raised high (3 Volts) and then either the DATA or /DATA line is lowered to 0 Volts to write a 1 or a 0 into the cell.

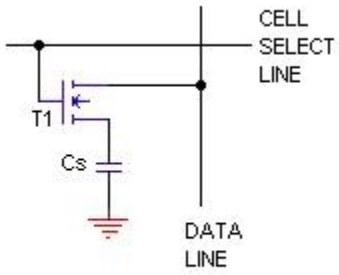
So static RAM is fast and expensive, and dynamic RAM is less expensive and slower.

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

Generally static RAM is used to create the CPU's speed-sensitive cache, while dynamic RAM forms the larger system RAM space.

DRAM (Dynamic RAM):

Just like static RAMs, the memory on a dynamic RAM memory chip is organized in a matrix formed by rows and columns of memory cells. The simplest type of dynamic RAM cell contains only one transistor and one capacitor. Whether a 1 or 0 is contained in a cell is determined by whether or not there is a charge on the capacitor. During a read operation, one of the row select lines is brought high by decoding the row address (low-order address bits). The activated row select line turns on the switch transistors for all cells in the selected row. This causes the refresh amplifier associated with each column to sense the voltage level on the corresponding capacitor and interpret it as a 0 or a 1. If it is more than 50 percent, it reads it as a 1; otherwise it reads it as a 0. The column address (high-order address bits) enables one cell in the selected row for the output. The read cycle is actually a read/write cycle. If a '1' is read then the cell is re-written to with a '1' to recharge it. However if a '0' is read no recharge is necessary.



DYNAMIC RAM

During this process, the capacitors in an entire row are disturbed. In order to retain the stored information, the same row of cells is rewritten by the refresh amplifiers. A write operation is done similarly except that the data input is stored in the selected cell while the other cells in the same row are simply refreshed. As a result of the storage discharge through pn-junction leakage current, dynamic memory cells must be repeatedly read and restored, this process is called memory refresh. The storage discharge rate increases as the operating temperature rises, and the necessary time interval between refreshes ranges from 1 to 100 ms.

The capacitor Cs discharges through the internal resistance of the NMOS transistor T1. Typically Cs = 0.2 pF and the internal resistance Rin = 10^{10} ohms, so:

$Cs \times Rin = 0.2 \times 10^{-12} \times 10^{10} \times 10^{3}$ ms = 2 ms

So the typical refresh time interval is 2 ms. Although a row of cells is refreshed during a read or write, the randomness of memory references cannot guarantee that every word

in a memory module is refreshed within the 2-ms time limit. A systematic way of accomplishing a memory refresh is through memory refresh cycles.

Memory Refresh Cycle

In a memory refresh cycle, a row address is sent to the memory chips, and a read operation is performed to refresh the selected row of cells. However, a refresh cycle differs from a regular memory read cycle in the following respects:

1. The address input to the memory chips does not come from the address bus. Instead, the row address is supplied by a binary counter called the refresh address counter. This counter is incremented by one for each memory refresh cycle so that it sequences through all the row addresses. The column address is not involved because all elements in a row are refreshed simultaneously.

2. During a memory refresh cycle, all memory chips are enabled so that memory refresh is performed on every chip in the memory module simultaneously. This reduces the number of refresh cycles. In a regular read cycle, at most one row of memory chips is enabled.

3. In addition to the chip enable control input, normally a dynamic RAM has a data output enable control. These two control inputs are combined internally so that the data output is forced to its high-impedance mode unless both control inputs are activated. During a memory refresh cycle, the data output enable control is deactivated. This is necessary because all the chips in the same column are selected and their data outputs are tied together. On the other hand, during a regular memory read cycle, only one row of chips is selected, consequently, the data output enable signal to each row is activated.

The memory cells have a whole support infrastructure of other specialized circuits. These circuits perform functions such as:

* Identifying each row and column (row address select and column address select)
* Keeping track of the refresh sequence (counter)
* Reading and restoring the signal from a cell (sense amplifier)
* Telling a cell whether it should take a charge or not (write enable)

Other functions of the memory controller include a series of tasks that include identifying the type, speed and amount of memory and checking for errors.

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

READ ONLY MEMORY (ROM)

There are five basic ROM types:

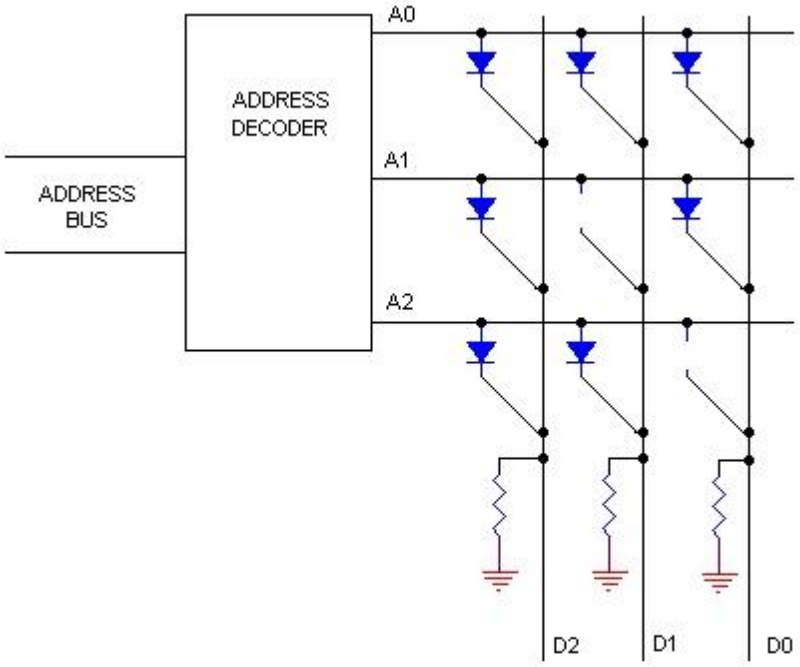
1. ROM - Read Only Memory
2. PROM - Programmable Read Only Memory
3. EPROM - Erasable Programmable Read Only Memory
4. EEPROM - Electrically Erasable Programmable Read Only Memory
5. Flash EEPROM memory

Each type has unique characteristics, but all types of ROM memory have two things in common:
Data stored in these chips is non-volatile -- it is not lost when power is removed.
Data stored in these chips is either unchangeable or requires a special operation to change.

ROM

A diode normally allows current to flow in only one direction and has a certain threshold, known as the forward breakover, that determines how much current is required before the diode will pass it on.
In silicon-based items such as processors and memory chips, the forward breakover voltage is approximately 0.6 volts.

By taking advantage of the unique properties of a diode, a ROM chip can send a charge that is above the forward breakover down the appropriate column with the selected row grounded to connect at a specific cell. If a diode is present at that cell, the charge will be conducted through to the ground, and, under the binary system, the cell will be read as being "on" (a value of 1). Iif the cell's value is 0, and there is no diode link at that intersection to connect the column and row. So the charge on the column does not get transferred to the row.



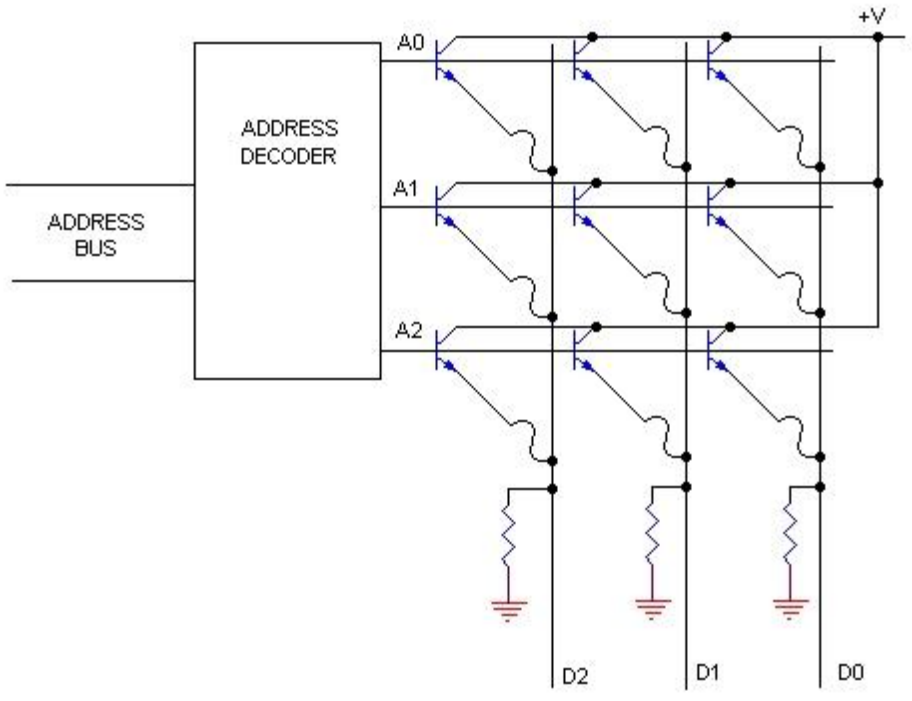
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

The way a ROM chip works necessitates the programming of complete data when the chip is created. You cannot reprogramme or rewrite a standard ROM chip. If it is incorrect, or the data needs to be updated, you have to throw it away and start over. Creating the original template for a ROM chip is often a laborious process. Once the template is completed, the actual chips can cost as little as a few cents each. They use very little power, are extremely reliable and, in the case of most small electronic devices, contain all the necessary programming to control the device.

PROM

Creating ROM chips totally from scratch is time-consuming and very expensive in small quantities. For this reason, developers created a type of ROM known as programmable read-only memory (PROM). Blank PROM chips can be bought inexpensively and coded by the user with a programmer.

PROM chips have a grid of columns and rows just as ordinary ROMs do. The difference is that every intersection of a column and row in a PROM chip has a fuse connecting them. A charge sent through a column will pass through the fuse in a cell to a grounded row indicating a value of 1. Sinc
e
all the cells have a fuse, the initial (blank) state of a PROM chip is all 1s. To change the value of a cell to 0, you use a programmer to send a specific amount of current to the cell. The higher voltage breaks the connection between the column and row by burning out the fuse. This process is known as
burning the PROM.



PROGRAMMABLE READ ONLY MEMORY (PROM)

Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

PROMs can only be programmed once. They are more fragile than ROMs. A jolt of static electricity can easily cause fuses in the PROM to burn out, changing essential bits from 1 to 0. But blank

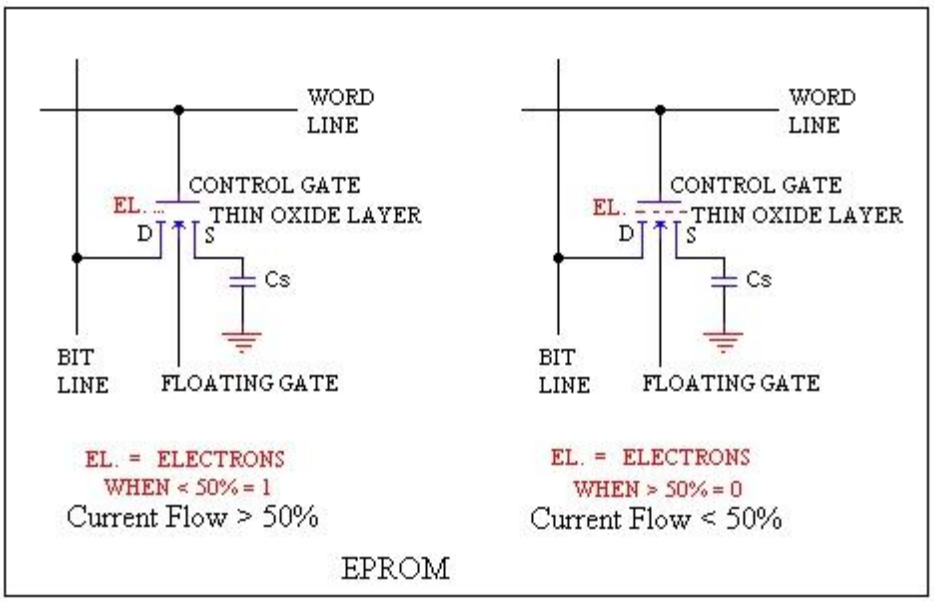
PROMs are inexpensive and are good for prototyping the data for a ROM before committing to the costly ROM fabrication process.

EPROM

Working with ROMs and PROMs can be a wasteful business. Even though they are inexpensive per chip, the cost can add up over time. Erasable programmable read-only memory (EPROM) addresses this issue. EPROM chips can be rewritten many times. Erasing an EPROM requires a special tool that emits a certain frequency of ultraviolet (UV) light. EPROMs are configured using an EPROM programmer that provides voltage at specified levels depending on the type of EPROM used.

The EPROM has a grid of columns and rows and the cell at each intersection has two transistors. The two transistors are separated from each other by a thin oxide layer. One of the transistors is known as the floating gate and the other as the control gate. The floating gate's only link to the row (wordline) is through the control gate. As long as this link is in place, the cell has a value of 1. To change the value to 0 requires a process called Fowler-Nordheim tunneling.

Tunneling is used to alter the placement of electrons in the floating gate. Tunneling creates an avalanche discharge of electrons, which have enough energy to pass through the insulating oxide layer and accumulate on the gate electrode. When the high voltage is removed, the electrons are trapped on the electrode. Because of the high insulation value of the silicon oxide surrounding the gate, the stored charge cannot readily leak away and the data can be retained for decades. An electrical charge, usually 10 to 13 volts, is applied to the floating gate. The charge comes from the column (bitline), enters the floating gate and drains to a ground.



Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

This charge causes the floating-gate transistor to act like an electron gun. The excited electrons are pushed through and trapped on the other side of the thin oxide layer, giving it a negative charge. These negatively charged electrons act as a barrier between the control gate and the floating gate. A device called a cell sensor monitors the level of the charge passing through the floating gate. If the flow through the gate is greater than 50 percent of the charge, it has a value of 1. When the charge passing through drops below the 50-percent threshold, the value changes to 0. A blank EPROM has all of the gates fully open, giving each cell a value of 1.
To rewrite an EPROM, you must erase it first. To erase it, you must supply a level of energy strong enough to break through the negative electrons blocking the floating gate. In a standard EPROM, this is best accomplished with UV light at a wavelength of 253.7 nanometers (2537 angstroms). Because this particular frequency will not penetrate most plastics or glasses, each EPROM chip has a quartz window on top of it. The EPROM must be very close to the eraser's light source, within an inch or two, to work properly.

An EPROM eraser is not selective, it will erase the entire EPROM. The EPROM must be removed from the device it is in and placed under the UV light of the EPROM eraser for several minutes. An EPROM that is left under too long can become over-erased. In such a case, the EPROM's floating gates are charged to the point that they are unable to hold the electrons at all.

EEPROMs and Flash Memory

Though EPROMs are a big step up from PROMs in terms of reusability, they still require dedicated equipment and a labor-intensive process to remove and reinstall them each time a change is necessary. Also, changes cannot be made incrementally to an EPROM; the whole chip must be erased. Electrically erasable programmable read-only memory (EEPROM) chips remove the biggest drawbacks of EPROMs.

In EEPROMs:

1. The chip does not have to removed to be rewritten.
2. The entire chip does not have to be completely erased to change a specific portion of it.
3. Changing the contents does not require additional dedicated equipment.

Instead of using UV light, you can return the electrons in the cells of an EEPROM to normal with the localized application of an electric field to each cell. This erases the targeted cells of the EEPROM, which can then be rewritten. EEPROMs are changed 1 byte at a time, which makes them versatile but slow. In fact, EEPROM chips are too slow to use in many products that make quick changes to the data stored on the chip.

Manufacturers responded to this limitation with Flash memory, a type of EEPROM that uses in-circuit wiring to erase by applying an electrical field to the entire chip or to predetermined sections of the chip called blocks. This erases the targeted area of the chip, which can then be rewritten. Flash memory works much faster than traditional EEPROMs because instead of erasing one byte at a time, it erases a block or the entire chip, and then rewrites it. The electrons in the cells of a Flash-memory chip can be returned to normal ("1") by the application of an electric field, a higher-voltage charge.

MEMORY TESTING

RAM TEST

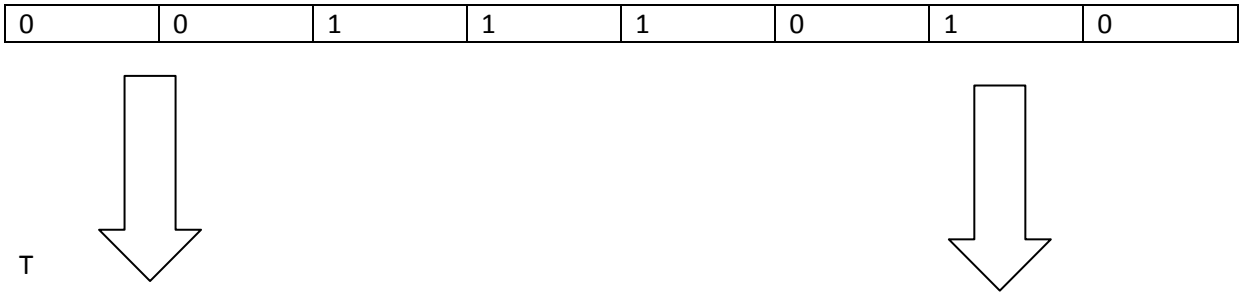
To test a block of RAM all locations must be capable of storing data and it must be possible to write into and read from each location.

Writing/reading random data will not fully test a RAM memory location as ground faults will read as a 0 and open circuit faults will read as a 1 and thus random data with a selection of zeros and ones will not fully test all bits in a RAM location.

For example writing the data $3A to a faulty RAM location (with the two upper bits shorted to ground and bit 1 open circuit) and then reading $3A from this location only tests that when read some bits are at zero and some are at one. Since in this extreme example the RAM location has the two upper bits permanently shorted to ground and bit 1 open circuit then writing/reading this data will not detect any fault since the upper bits will always read as zeros and bit 1 will always read as 1 and hence the data will read as $3A since the other positions store the correct bits when written to.

This faulty memory location will also read as $3A when for example data $F8 is written to it.

In Binary $3A - 00111010



To fully test a RAM location it is necessary to first ensure that all locations can write/read $00 and then ensure that all locations can write/read $FF i.e that all bits in all locations can store a 0 ***and*** a 1.

Refer to SECTION 13 (Past exams and solutions) for a flowchart for testing a block of RAM

ROM TEST

Since ROM locations contain data which should never change then a simple test is to add the contents of all ROM locations in the block under test together. The result called a ***checksum*** should always be the same. Since this checksum can be a large number to ensure 100% accuracy the whole checksum should be verified. If only the lower byte is verified then a small possibility exists of an error not been detected.

For example if when adding together the contents of a 10 k block of ROM the result was $45AF6DEE and the only check made was to ensure that the result was $6DEE (the lower 16 bits of the result) then the possibility exists that some locations were faulty. If the correct checksum should be $F5FA6DEE then only checking the lower bits will not detect a fault.

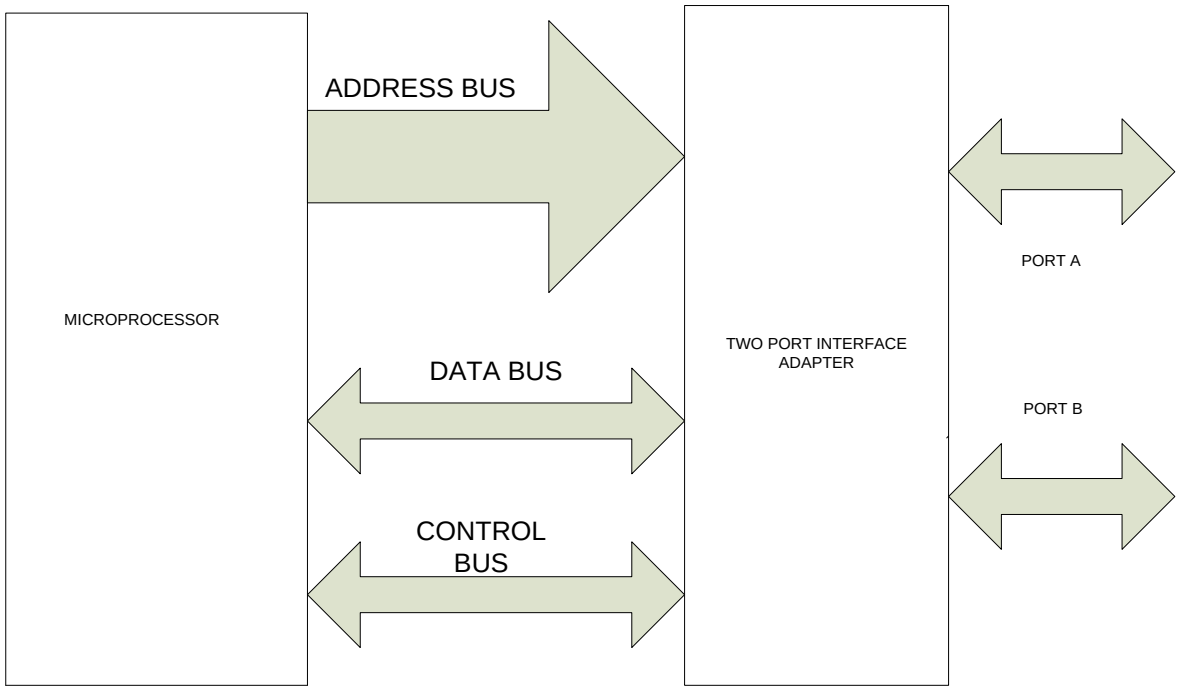
The size of the block of ROM will determine the size of the checksum.

Refer to SECTION 13 (Past exams and solutions) for a flowchart for testing a block of ROM

SECTION 10

PARALLEL COMMUNICATIONS

In the simple 8 bit microprocessor under consideration such as the 6800 family there are no dedicated input output (I/O) lines and so in order to use any I/O device a suitable interface device is required. This interface device must be connected to the microprocessor via the address, data and control buses and in so doing is hard-wired to a specific address range. This is shown in the diagram below where a two port interface device is connected to the microprocessor. In the case of the 6800 family of microprocessors this device is known as a Peripheral Interface Adapter (P.I.A).



This type of I/O is called memory mapped I/O because the I/O device is allocated a memory location and thus is treated as any normal location – the same instructions such as LOAD and STORE are used to input and output data as are used to read/write to memory.

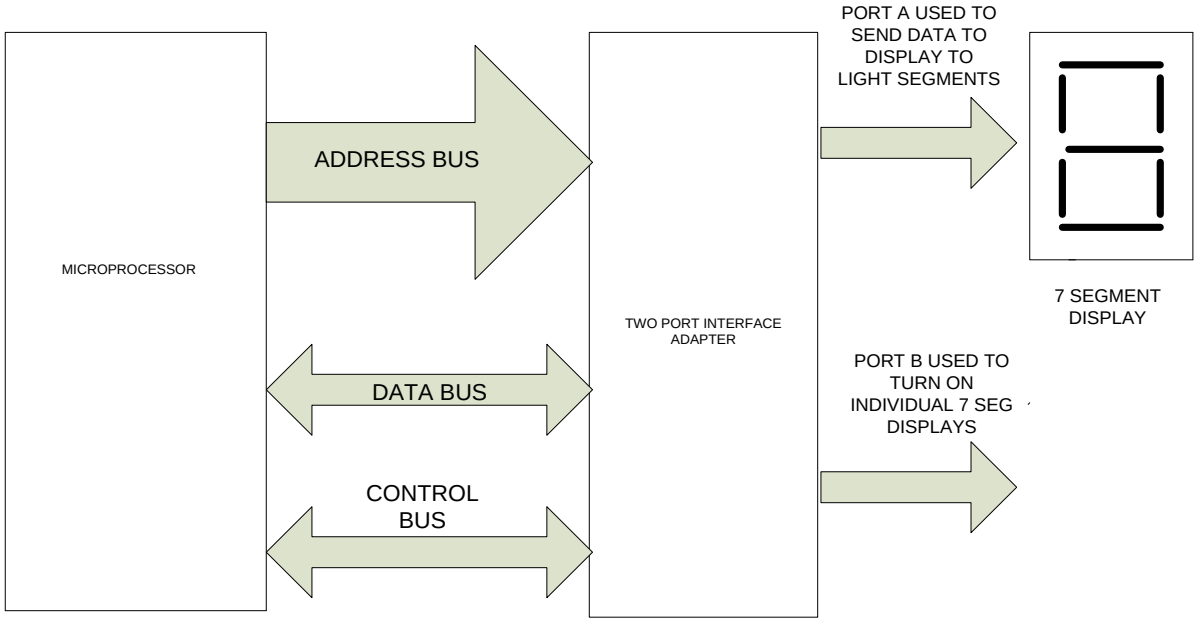
Another system in use is port mapped I/O where the microprocessor has dedicated I/O lines and uses special instructions such as IN and OUT to communicate with the I/O device. In this system the number of I/O devices that can be connected to the microprocessor is limited by the number of I/O pins on the microprocessor.

In both cases each I/O line can be configured as either an input or an output.

As an example consider the case of interfacing a single seven segment display to a 6800/6809 microprocessor:

In the diagram below Port A is used to send segment data to the display and depending on the type of display used and its connections either a '1' or a '0' will light a segment. If a '1' is used then outputting the hexadecimal data $7F (0111 1111) will light all segments and display an 8.

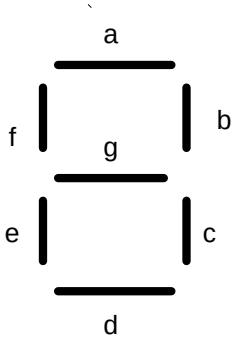
The data output at Port B determines which (if more than one) of the seven segment displays are to be turned on.



TYPICAL SEVEN SEGMENT DISPLAY

The segments are labelled a,b,c,d,e,f and g as shown in the seven segment display diagram below.

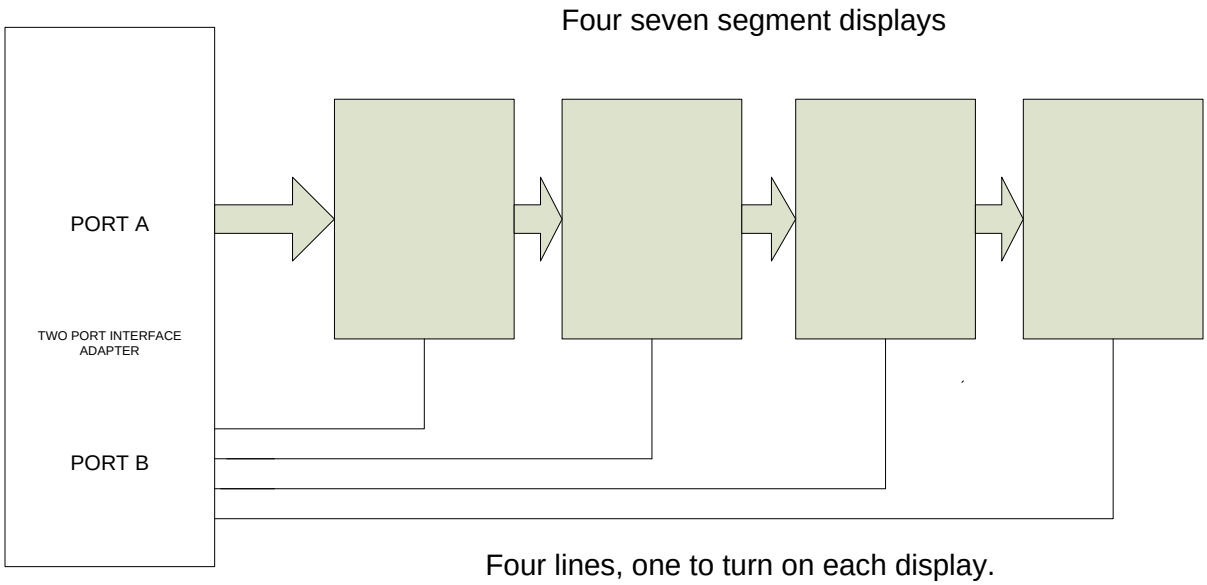
By lighting different segments, under program control, characters/digits can be displayed.



Thus it is possible, by lighting the relevant segments, to display a range of decimal and hexadecimal numbers and alphanumeric characters.

DISPLAY MULTIPLEXING

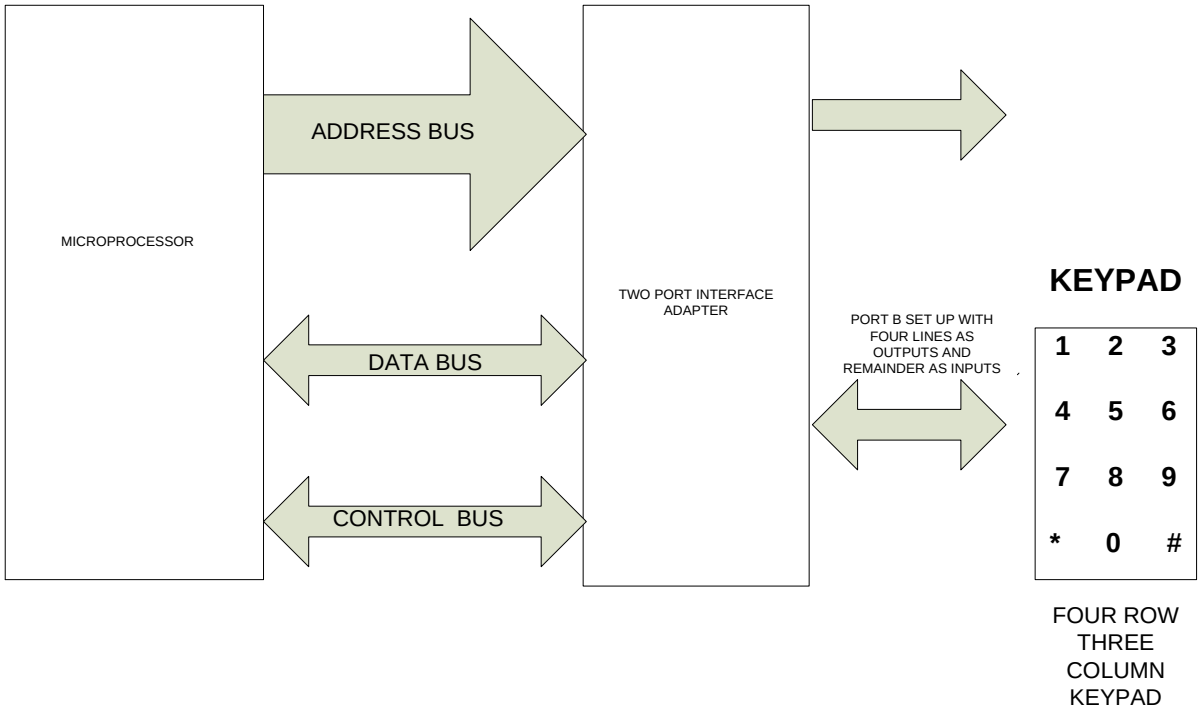
As an example consider the case of interfacing a number of seven segment displays to a 6800/6809 microprocessor. This is known as display multiplexing:



Here the data output on Port A to light segments is common to the four displays but only one display is turned on (under program control) at a time. The speed of the microprocessor ensures that all displays appear to be on at all times and thus four (or more) characters or digits can be displayed simultaneously.

This method reduces the power consumption of the display array but the trade off is that the displays appear dimmer.

As an example consider the case of interfacing a simple keypad to a 6800/6809 microprocessor:



In this case only one port (Port B) is used and the 4 lower lines in this port are set up as output lines and the 4 upper lines as inputs (of which only three are used).

To scan a keypad it is necessary to output a '1' to each row in turn and look for a '1' appearing on one of the three column input lines. When a key is pressed it interconnects a row with a column and by knowing which row and which column the key pressed can be determined.

The keypad scan routine can be run at regular intervals which in microprocessor speed times would be every millisecond or so and this can be done by a timer interrupt calling a keypad scan subroutine or some other suitable method. Switch debouncing may be necessary which simply means a short delay incorporated into the scan routine to ensure that only one key press is considered and any key bounce is ignored.

SECTION 11

SERIAL COMMUNICATIONS

Serial transmission is often preferred over parallel transmission, even though it has a lower transfer rate, due to its simplicity, low cost and ease of use. Many peripherals also do not require the high data rates of a parallel interface.

Baud rate

The number of changes/symbols transmitted, per second. When there are only two states this is equal to the Bit rate.

Bit rate

The number of bits transferred per second.

Data rate

The rate at which meaningful information is sent - the bit rate less the overhead of start and stop bits.

Asynchronous Serial Data Transmission

The asynchronous serial interface is so called because the transmitted and received data are not synchronised over any extended period of time and therefore no special means of synchronising the clocks at the transmitter and receiver is necessary. The fundamental problem lies in how to split the data stream into individual bits and how to then reconstruct the original data. The format of the data on a serial data link is in fact simple, and is shown in Figure 1 below. Data is grouped and transferred in characters, where one character is a unit comprising 7 or 8 bits of information plus 2 to 4 control bits. The idle state is referred to as the mark level and traditionally corresponds to a logical 1 level. A character is transmitted by placing the line in the space level (logical 0) for one period T, then the information is sent bit by bit, with each bit T seconds long, then the transmitter calculates the parity bit and transmits it and finally one or two stop bits are sent by returning the line to mark level.

Format of Asynchronous Serial Data

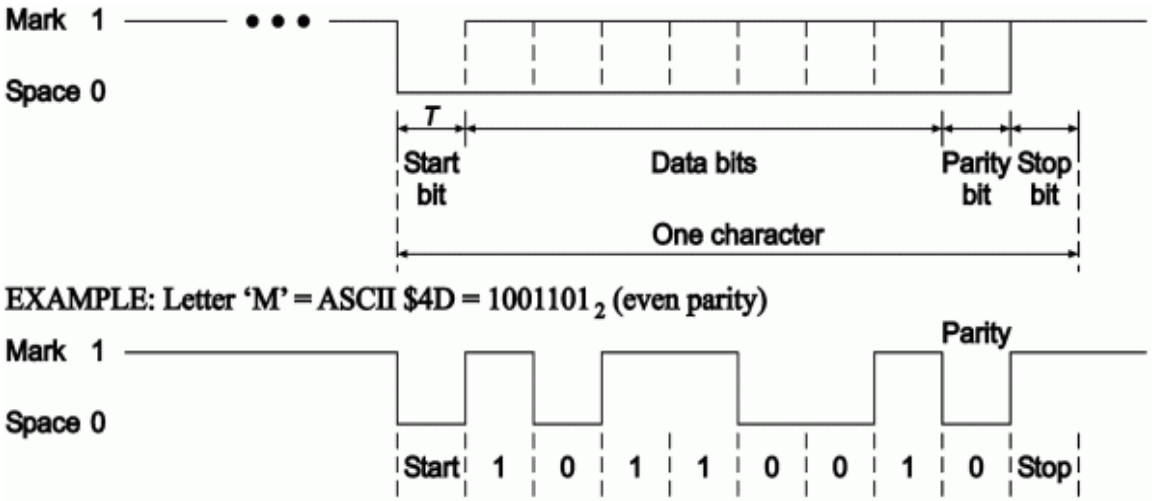


Figure 1

The circuit below will generate serial asynchronous data with one stop bit, a parity bit and a start bit.

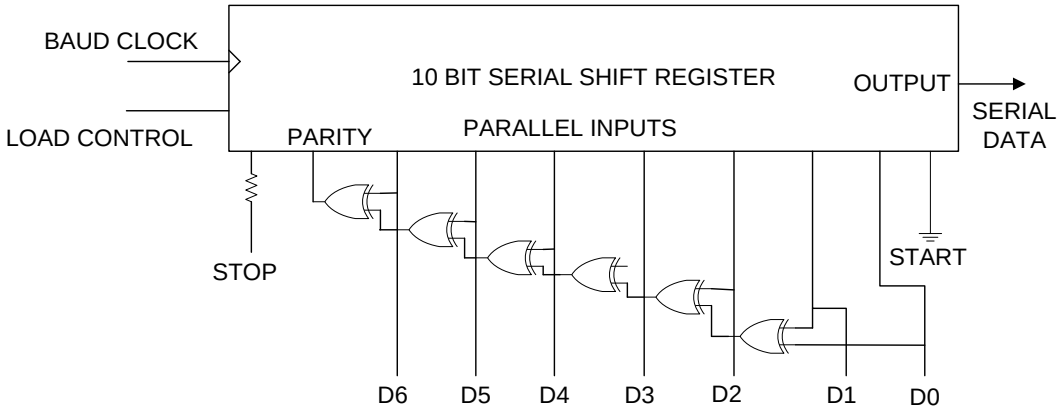


Figure 2

This circuit (Figure 2) is based on a 10 bit parallel to serial shift register. The data to be framed with start and stop bits is presented in parallel to the shift register. The exclusive OR gates generate the parity bit according to the number of ones in the data. A load control signal loads the data (with parity bit, low start bit and high stop bit) into the register which is then clocked out serially at a rate determined by the baud clock generating a serial signal similar to that shown in Figure 1 above.

The data word length may be 7 or 8 bits with odd, even, or no parity bits plus either 1 or 2 stop bits. This allows for 12 different possible formats for serial transmission. Also, there are at least 7 different commonly used values for the bit period T. Thus, connecting two devices via a serial link may be difficult due to all the available options.
At the receiving end, the receiver monitors the link, looking for the start bit, and once detected, the receiver waits until the end of the start bit and then samples the next N bits at their centres using a locally generated clock.

The circuit below (Figure 3) is used to detect a valid start bit on the serial line. A start bit on the line (which is a high to low transition remaining low for a set period) will generate (via the one shot) a clock pulse for the negative edge triggered D type flip flop at the end of the one shot's pulse which will in turn cause the valid start output to go high provided the start bit is still low. If this is not the case there will be no valid start bit.

This is illustrated in the waveforms in Figure 4 below where noise on the line will not generate a valid start but a start bit on the line will (generate a valid start).

Circuit to detect a valid start bit on a serial line:

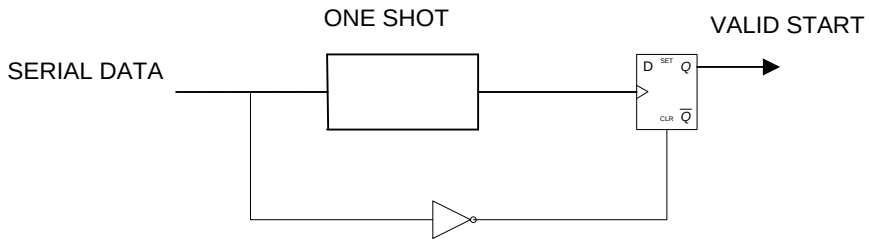


Figure 3

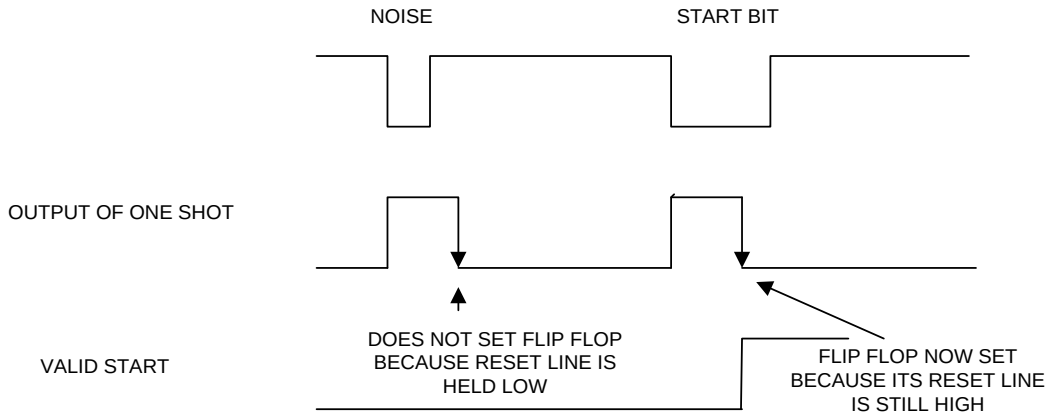


Figure 4

Once the character has been assembled from the received bits, the parity is checked, and if the calculated parity does not agree with the received parity bit, a parity error flag is set to indicate a transmission error. The most critical aspect is the receiver timing. The falling edge of the start bit triggers the receiver's local clock, which samples each incoming bit at its nominal centre. Suppose the receiver clock waits T/2 seconds from the falling edge of a start bit and samples the incoming data every T seconds thereafter until the stop bit has been sampled. Let us assume that the receiver clock is running slow, so that a sample is taken every T+dt seconds. The first bit of the data is thus sampled at (T+dt)/2 + (T+dt) seconds after the falling edge of the start bit. The stop bit is thus sampled attime (T+dt)/2 + N(T+dt), where N is the number of bits in the character following the start bit. The total accumulated

error in sampling is thus (T+dt)/2 + N(T+dt) – (T/2 + NT), or (2N = 1)dt/2 seconds. This situation is shown in Figure 5 below.

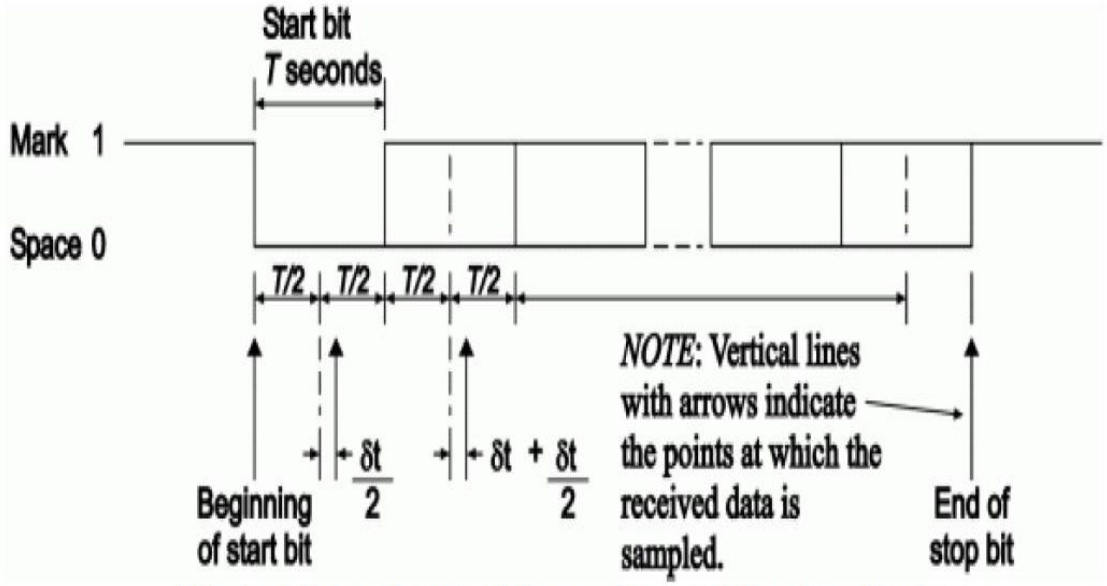


Figure 5

For correct operation, the stop bit is sampled within T/2 seconds of its centre. Thus, if N=9 for a 7-bit character with one stop bit and one parity bit, the maximum permissible error is 100/19 = 5%. Fortunately, today's clocks are all crystal controlled and the error between two clocks of the same frequency is often less than a fraction of a percent.
The most obvious disadvantage of asynchronous serial transfer is the need for start, stop and parity bits for each transmitted character. If 7-bit characters are used, the overall efficiency is only 7/(7+3) = 70%. Another problem is when asynchronous transfer is used to, for example, dump binary data onto a storage device: If the data is arranged in 8-bit bytes and all 256 values represent valid binary data it is difficult to embed control characters (e.g. tape start or stop) within the data stream because the same character must be used both for pure data and control purposes.

Synchronous Serial Data Transmission

The type of asynchronous serial data link described in previous sections is widely used to link processors to relatively slow peripherals such as printers and terminals. Where information must be transferred, for example, between individual computers in a network, synchronous serial data transmission is more popular. In synchronous serial data transmission, the information is transmitted continuously without gaps between adjacent groups of bits. Note that synchronous data links are often used to transmit entire blocks of data instead of ASCII-encoded characters.
As this type of link involves long streams of data, the clocks at the receiving and transmitting end must be permanently synchronized. Of course, one could simply add a clock line to the link where the transmitter's clock signal is passed to the receiver. However, this requires an

additional line and is thus an unpopular choice. A better solution is to encode the data in such a way that the synchronizing signal is included in the data signal. The figure below (Figure 6) shows one of the many methods which may be used. In this case the data is phase-encoded (or Manchester encoded) by combining the clock signal with the data signal. A logical one is thus represented by a positive transition in the centre of the bit and a logical zero by a negative transition. At the receiver, the data signal may easily be split into the clock and pure data components. Integrated circuits that perform this modulation and demodulation are readily available.

A Phase-Encoded Synchronous Serial Bit Stream

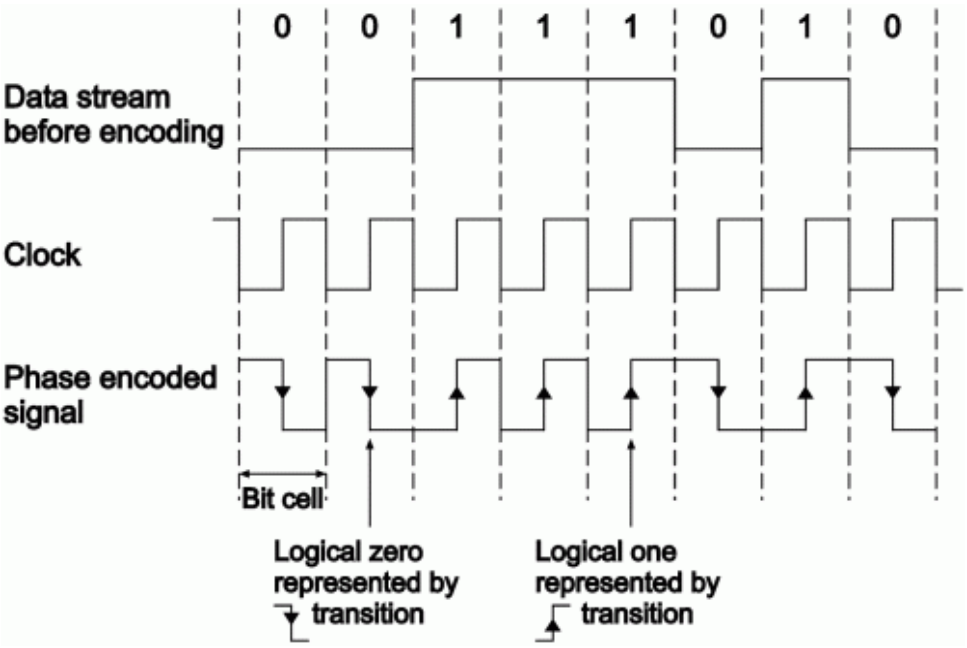


Figure 6

Having divided the incoming data stream into individual data elements (i.e. bits), the next step is to group the bits into meaningful units. The incoming data must be examined for recognizable bit groups which signify the beginning of a block of data, the end of it or some other control character.

Parity

There may well be noise on a communications line, and it is helpful to have some check that the correct information has arrived. One common test is Parity. Send a parity bit set so that the number of 'ones' sent (data + parity) is odd.
Parity is normally taken as odd, because a single pulse on the line, taken as a start bit, records as a bad byte.
The parity check will detect one erroneous bit in each byte. There are more serious methods of encoding data that can send messages down noisy lines, and recover

from erroneous bits.

A Universal Asynchronous Receiver Transmitter (UART)

Serial information is normally transmitted and received by a Universal Asynchronous Receiver Transmitter (UART). The programming interface for a UART usually has four registers:

Transmit control - baud rate, parity, send now

Transmit data - the byte to be sent

Receive control - byte available, parity check,

Receive data

SECTION 12

DT008/2 MICROPROCESSOR SYSTEMS 1

SAMPLE PROBLEMS:

1 Using the relative addressing mode of the 6800 microprocessor what are the maximum offsets available?

A branch if greater than or equal to (BGE) instruction, of code and offset, is located at $E020 and $E021 respectively.

Write down the contents of the condition code register that would enable this branch to occur and calculate the destination address when the branch offsets are:

(i) FC;

(ii) A0;

(iii) 7C.

2 The instruction *Load Index Register* (LDX*)* is contained in the instruction set for the 6800 microprocessor and can be executed using four addressing modes. Describe the operation of the instruction for each mode giving a suitable example in each case.

3 A 6800 microprocessor executes the following instructions:

```
LDX   #$E030
LDAA   #$7A
LDAB   0,X
ABA
STAA   $E050
```

What are the four addressing modes used?

Explain what each instruction does and what is the end result of the execution of these instructions?

How many memory locations would these instructions require when converting to machine code and what are the machine codes for these instructions.

Assuming that the memory location $E030 contains $8C write down the hexadecimal contents of the condition code register after the execution of each of the instructions above and explain the answer

4 What does the following program do?

```
START: LDX   #$F0FF
LOOP:  DEX
       BNE   LOOP
       RTS
```

Convert the above assembly language program to 6800-microprocessor machine code. The program is to start at memory location $E000.

If the 6800 microprocessors clock operates at 2.5 MHz determine the delay introduced by the above program.

SOLUTIONS TO SAMPLE PROBLEMS

1 Since the 6800 uses 8 bit twos complement numbers as the offset in relative addressing the maximum offsets available are 127 steps forward (+127 = 01111111) and 128 steps backwards (-128 = 10000000). In hex the offset range is $7F - $80.

According to the instruction set BGE tests the V (overflow) bit and the N (negative result) bits and will branch under the following conditions:

If the result is not negative and no overflow has occurred then the result must be ≥ 0.

If the result is negative and overflow has occurred then the result is ≥ 0.

Otherwise the result is less than zero.

CCR	H	I	N	Z	V	C	
	0	0	0	0	0	0	OR
	0	0	1	0	1	0	

(i) The destination address will be -4 steps back ($FC) and since the program counter is at $E022 the destination address is

$$\$E022 - \$04 = \$E01E$$

$FC = 11111100, inverting = 00000011, add 1 = 00000100 = $04

Hence $FC represents -04_{10}.

(ii) The destination address will be -96 steps back ($A0) and since the program counter is at $E022 the destination address is

$$\$E022 - \$60 = \$DFC2$$

Note that $96_{10} = \$60$ and in order to determine what the twos complement number $A0 represents reverse the twos complement process on $A0.

$A0 = 10100000, inverting = 01011111, add 1 = 01100000 = $60

$60 = 01100000 = 64+32 = 96_{10} (Note that the subscript 10 means decimal) hence $A0 represents -96_{10}.

(iii) The destination address will be 124 steps ahead ($7C) and since the program counter is at $E022 the destination address is

$$\$E022 + \$7C = \$E09E$$

2 The four addressing modes that this instruction can use are immediate, extended, indexed and direct.

Example of instruction using immediate addressing:

LDX #$1234

This will immediately put the 16 bit number $1234 into the (16 bit) index register.

Example of instruction using direct addressing:

LDX $64

This will load the index register with the contents of the two memory locations $0064 and $0065. (The contents of $0064 are placed in the lower byte of the IX and the contents of $0065 in the upper byte). The direct addressing mode of the 6800 can only address the first 256 memory locations $0000 to $00FF and is a two byte instruction.

Example of instruction using extended addressing:

LDX $1234

This will load the IX with the contents of the two memory locations $1234 and $1235. The extended addressing mode can address all memory locations ($0000 to $FFFF).

Example of instruction using indexed addressing:

LDX 0,X

This will load the IX with the contents of the two memory locations pointed to by the IX (The pointed to location and the next location).

For example if the IX contains $1234 then this instruction will load the index register with the contents of the two memory locations $1234 and $1235.

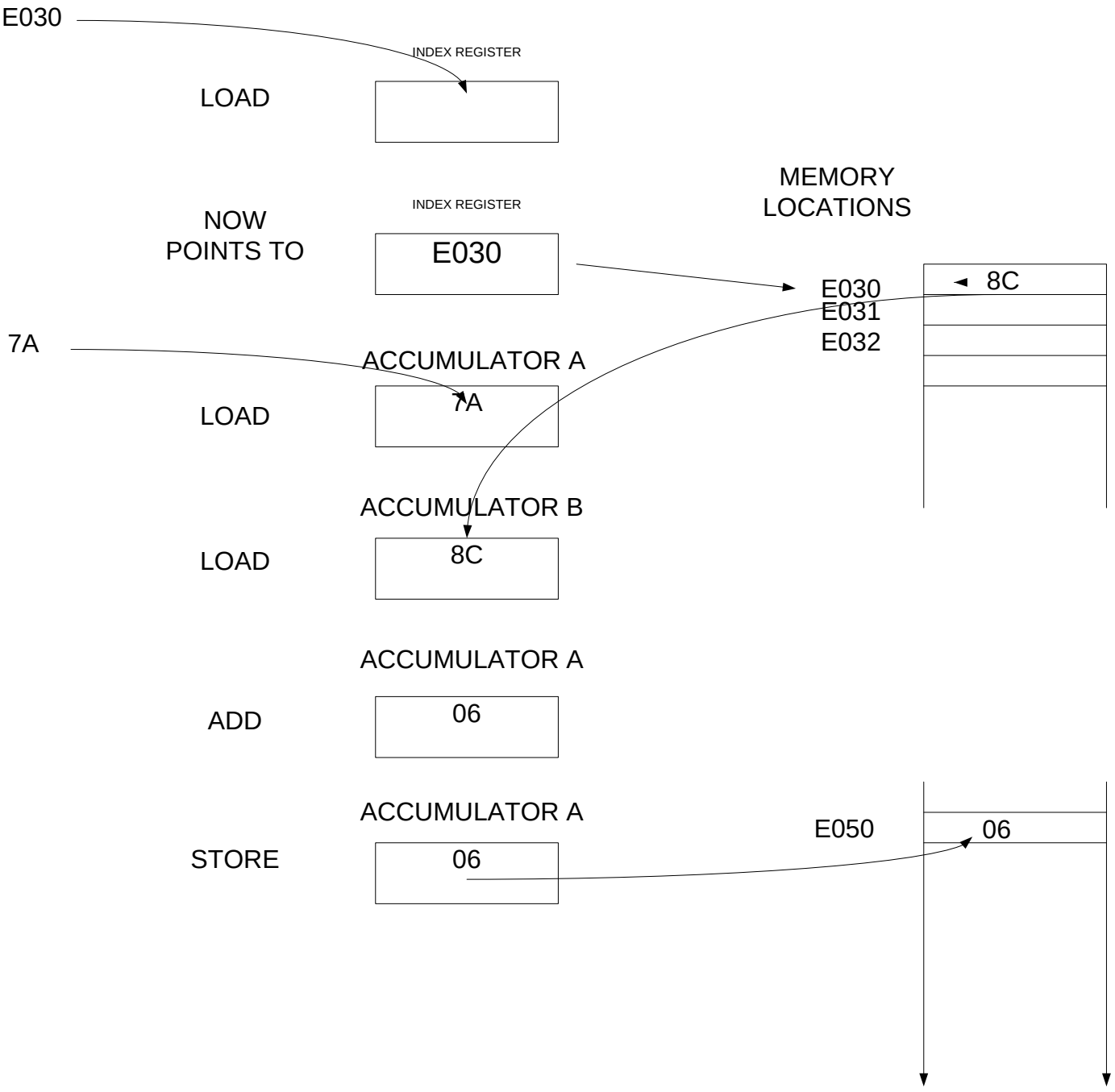
Indexed addressing has an offset and this two btye number can offset the pointed to location by 0 – 256 ($00 to $FF). For example LDX $FF,X will load the IX with the contents of the memory locations (IX) + $FF and (IX) +$FF +$01.

3 The addressing modes are as follows:

```
LDX   $E030   Extended
LDA   #$7A    Immediate
LDB    0,X    Indexed
ABA          Implied
STA    $E050  Extended
```

These instructions load the X index register with $E030 and the accumulator A with $7A. The data stored in location $E0A0 is then placed in accumulator B and the accumulator contents are added together and the result stored in $E050.

ASSEMBLER INSTRUCTIONS			MACHINE CODE	NO OF MEMORY LOCATIONS
LDX	$E030	Extended	CE E0 30	3
LDA	#$7A	Immediate	86 7A	2
LDB	0,X	Indexed	E6 00	2
ABA		Implied	1B	1
STA	$E050	Extended	B7 E050	3



ADDING $7A AND $8C

7A = 0111 1010

+8C = + 1000 1100

= 1]000 0110

This addition will set the half carry, and the full carry bits in the condition code register (CCR). The result is not negative and not zero and no (two's complement) overflow has occurred so the CCR will be

H	I	N	Z	V	C
1	1	1	x	0	0	0	1

Note x = don't care state (the interrupt bit may or may not be set) and the two most significant bits are not used so default to a 1.

4 This is a delay (sub)routine and the program loads a register (in this case the X index register) with a number ($F0FF) and then counts it down to zero. After every decrement a check is made to see if the Z (zero) bit in the CCR is set (indicating that the result is zero) and if not then a conditional branch occurs to repeat that step.

If the Z bit is set then the program ends (or returns to the main calling routine if ir is a subroutine).

The time it takes to count down to zero is the delay and this in turn depends on the number loaded into the register.

ASSEMBLER INSTRUCTIONS			MACHINE CODE		
START:	LDX	#$F0FF	START: EE00	CE F0 FF	
LOOP:	DEX		LOOP: EE03	09	
	BNE	LOOP	EE04	26 FD	
	RTS		EE06	39	

Note that the conditional branch instruction will (if condition is met) require program execution to continue at LOOP which means that it must go back 3 steps (since the program counter is always one step ahead). Thus the offset must be a two's complement number representing -3.

+03 = 00000011

Invert = 11111100

Add 1 = 11111101 = -3 = $FD

To determine the delay it is necessary to count the number of clock cycles that are required to complete the program. From the instruction set the clock cycles for each instruction are as follows:

		CLOCK CYCLES
START: LDX #$F0FF	START: EE00 CE F0 FF	3

LOOP:	DEX	LOOP:	EE03	09	4
BNE	LOOP	EE04	26 FD		4
RTS		EE06	39		5

Since the program executes the DEX and BNE instructions $F0FF times the total no of clock cycles is

$$4 + ((4 + 4).(\$F0FF)) + 5 = TOTAL$$

$F0FF = 61695_{10} so TOTAL = 493569.

The clock frequency is 2.5 MHz so each clock cycle is 0.4 µS and hence the total delay is 0.1974 Seconds.

SECTION 13

PAST EXAM PAPERS AND SOLUTIONS

SEMESTER 1 EXAMS 2011

1 (a) A 6809 microprocessor during program execution can use either of the two instructions:

BRA $A0 or

JMP $E050

and both of these instructions result in program execution continuing at $E050.

What are the differences in operation between using the branch or the jump instruction?

Where in program memory would each of these instructions reside, if used, and what are the machine codes for these instructions?

[13.3 marks]

(b) Write a short note outlining the functions of each of the following internal registers of the 6800 microprocessor.

(i) The condition code register.

(ii) The program counter.

(iii) The index register.

(iv) The A and B accumulators.

[10 marks]

(c) Using the relative addressing mode of the 6800 microprocessor what are the maximum offsets available?

A branch if greater than or equal to (BGE) instruction, of code and offset, is located at $E020 and $E021 respectively.

How does the microprocessor determine whether to branch or not?

Write down the contents of the condition code register that would enable this branch to occur and calculate the destination address when the branch offset is $7C.

[10 marks]

2 (a) Write a short note outlining the procedure for testing a block of read only memory (ROM).

[10.3 marks]

(b) A kilobyte block of ROM starting from $E000 is to be tested. The checksum is $EDFC. If this is correct then data $AA is to be stored in location $E100 otherwise data $BB is to be stored in location $E100.

With the aid of a flowchart demonstrate a method of testing this block of ROM.

[12 marks]

(c) Write an assembly language program to implement the flowchart in part (b). Assume the origin is at $ED80. Use one of the microprocessor instruction sets attached.

[11 marks]

3 (a) How many address lines does a 6809 microprocessor have and how many memory locations can it address?

[3 marks]

(b) Write a short note outlining the need for additional address decoding circuitry in a microcomputer system which utilizes the 6809 microprocessor.

[7 marks]

(c) Give the truth table and logic symbol for a 3 to 8 line decoder of the type commonly used in address decoding circuitry in microcomputer memory systems.

[7 marks]

(d) Show by means of a block diagram of the address decoding system how one of the above decoders could be used to implement the memory map as shown in Figure 1 below. Indicate the size of each memory block.

[16 .3marks]

MEMORY RANGE	MEMORY TYPE
0000 – 0FFF	RAM1
2000 – 2FFF	RAM2
4000 – 4FFF	ROM1
7000 – 7FFF	ROM2

Figure 1

4 (a) Write a short note outlining the differences between serial and parallel transmission of data.

[8 marks]

(b) Draw a block diagram of a microprocessor system which uses parallel transmission of data to display digits on a typical seven segment display.

Show the microprocessor, parallel device, display device, control, address and data lines.

[10 marks]

(c) Draw a block diagram of a microprocessor system which uses serial transmission of data to send data to a printer. Sketch the output digital waveform if the serial device is set to use one start bit, two stop bits, even parity and the transmitted data is $8C.

[15.3 marks]

SEMESTER 1 SUPPLEMENTAL 2011

1 (a) A 6800 microprocessor during program execution can use either a branch or a jump instruction and both of these instructions can result in program execution continuing at the same location.

Write a short note on the differences in operation between using the branch or the jump instruction and clearly explain how the use of either in a program can result in program execution continuing at the same location.

[10 marks]

(c) Outline the differences between the 6800 and 6809 microprocessors under the following headings:

(i) Internal registers,

(ii) Instruction set,

(iii) Clock speed and,

(iv) Pin layout.

[13.3 marks]

(c) Using the relative addressing mode of the 6800 microprocessor what are the maximum offsets available?

A branch if greater than or equal to (BGE) instruction, of code and offset, is located at $E020 and $E021 respectively.

How does the microprocessor determine whether to branch or not?

[10 marks]

2 (a) Write a short note outlining the difference in procedures for testing a block of random access memory (RAM) and read only memory (ROM).

[10.3 marks]

(b) Give a general flowchart for testing a block of RAM.

[11 marks]

(d) Give a general flowchart for testing a block of ROM.

[12 marks]

3 (a) How many address lines does a 6809 microprocessor have and how many memory locations can it address?

[3 marks]

(b) Write a short note outlining the need for additional address decoding circuitry in a microcomputer system which utilizes the 6809 microprocessor.

[7 marks]

(c) Give the truth table and logic symbol for a 3 to 8 line decoder of the type commonly used in address decoding circuitry in microcomputer memory systems.

[7 marks]

(e) Give a block diagram of the memory element of a 6800 based microcomputer system which uses two 2k byte blocks of RAM and two 4K byte blocks of ROM. The diagram should show the microprocessor, the four memory blocks, the address decoding element and all interconnections.

[16.3 marks]

4 (a) Write a short note outlining how input/output devices such as 7 segment displays and keypads are connected to a 6800 based microcomputer system.

[8 marks]

(b) Draw a block diagram of a microprocessor system which uses parallel transmission of data to display digits on a typical seven segment display.

Show the microprocessor, parallel device, display device, control, address and data lines.

[13.3 marks]

(c) Draw a block diagram of a microprocessor system which uses serial transmission of data to send data to a printer. Sketch the output digital waveform if the serial device is set to use one start bit, two stop bits, even parity and the transmitted data is $8C.

[12 marks]

PAST EXAM PAPER SOLUTIONS

SEMESTER 1 2011 SOLUTIONS

Q1 (a) The branch instruction uses relative addressing and where it branches to is dependant on where the branch instruction is located in program memory. The jump instruction will always cause program execution to continue at the specified address regardless of where in program memory the instruction is located.

Since execution continues at $E050 and the branch instruction causes a backward branch of $60 (96) steps from the current program counter then the program counter must be at $E050 +$60 = $E0B0.

Since the program counter is always one step ahead the branch instruction must be located as follows:

$E0AE	20	Machine code for BRA instruction
$E0AF	A0	

And the jump instruction as follows:

$E0AD	7E	Machine code for JMP instruction
$E0AE	E0	
$EOAF	50	

[13 marks]

(b) (i) The condition code register contains flags which indicate the results of the last arithmetic or logical instruction executed by the microprocessor. These flags are as follows:

H Half carry

I Interrupt

N Negative result

Z Zero result

V Twos complement overflow

C Full carry generated

(ii) The Program Counter (PC) always contains the address of the next instruction to be executed. This is a 16 bit register and certain instructions such as jump or branch instructions can modify the contents of the PC so that program execution continues at a new location.

(iii) The Index Register (IX) is a 16 bit register which is used in Indexed Addressing. In this addressing mode the contents of the IX point to or specify the memory location where data is to be stored to or taken from.

(iv) The two accumulators (8 bit) are general purpose registers used for the manipulation of data during program execution.

[10 marks]

(c) The maximum offsets available are 127 steps forward and 128 steps backwards. In hex the offset range is $7F - $80.

According to the instruction set BGE tests the V (overflow) bit and the N (negative result) bits and will branch under the following conditions:

If the result is not negative and no overflow has occurred then the result must be ≥ 0.

If the result is negative and overflow has occurred then the result is ≥ 0.

Otherwise the result is less than zero.

CCR H I N Z V C

0 0 0 0 0 0 OR

0 0 1 0 1 0

The destination address will be 124 steps ahead ($7C) and since the program counter is at $E022 the destination address is

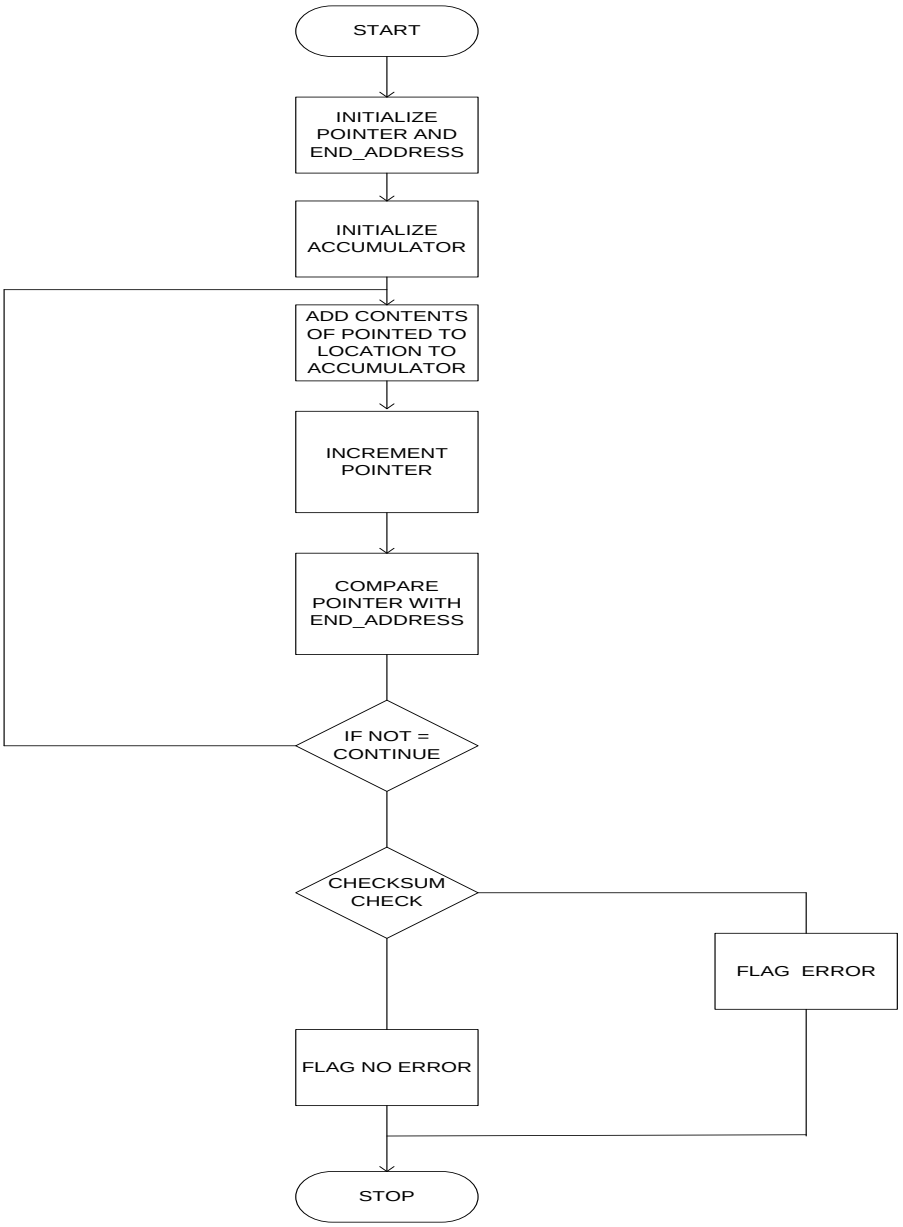
$$\$E022 + \$7C = \$E09E$$

[10 marks]

Q2 (a) To test ROM it is necessary to add together the contents of all the ROM locations and ensure that the result is always the same. This result is called the CHECKSUM. If the checksum is different it means a ROM location is faulty. If the checksum is a 32 or 16 bit number then the addition process must take account of carries generated.

[10 marks]

(b) Flowchart for testing ROM.



[12 marks]

(c) Assembly Language Program

```
                ORG $ED80
        POINTER EQU $EE00
END_ADDRESS EQU $E100
CHECKSUM EQU $EDFE
GOOD        EQU $AA
BAD         EQU $BB
RESULT      EQU $E100

START: CLRA
        CLRB
            LDX #POINTER
CONT:  ADDA 0,X
            BCC SKIP
            INCB
 SKIP   INX
            CMPX #END_ADDRESS
            BNE CONT
            CMPD #CHECKSUM
```

```
        BNE ERROR
        LDA #GOOD
        STA  RESULT
        RTS

ERROR:LDA #BAD
        STA RESULT
        RTS
```

[11 marks]

Q3 (a) The 6800 series of 8 bit microprocessors have 16 address lines and this gives a capacity of addressing up to 16^16 = 65536 locations

[3 marks]

(b) A microprocessor system requires different types and sizes of memory For example it needs ROM for monitor programs, RAM for data storage, EPROM for development and in the case of the 6800 series allocated memory space for memory mapped I/O purposes. Hence there is a need for additional address decoding circuitry to select each of the different memory devices attached to it each with a different address range.

[7 marks]

(c) Logic Symbol for 3 to 8 line decoder:

Truth Table for 3 to 8 line decoder

INPUTS OUTPUTS

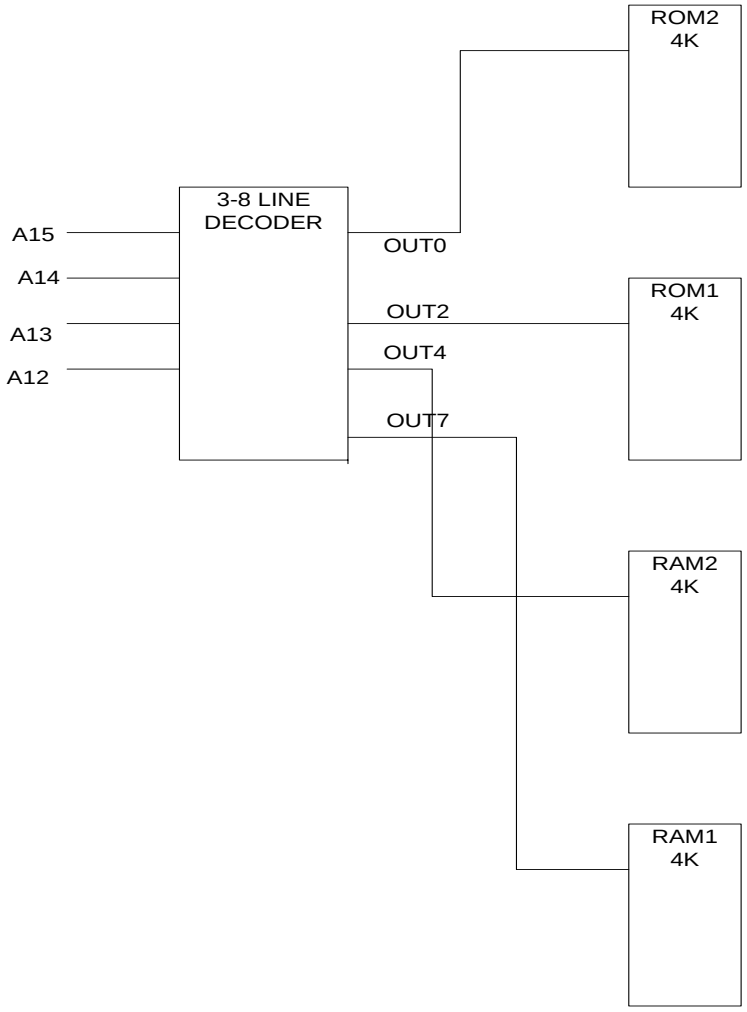
S2	S1	S0	O7	O6	O5	O4	O3	O2	O1	O0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

[7 marks]

RANGE	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0FFF	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
2000	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2FFF	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1
4000	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4FFF	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1
7000	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
7FFF	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

From the above table it can be seen that A_{15} can be used as the (active low) enable input, and A_{12}, A_{13} and A_{14} can be used to select one of the four memory elements as shown in the block diagram below.



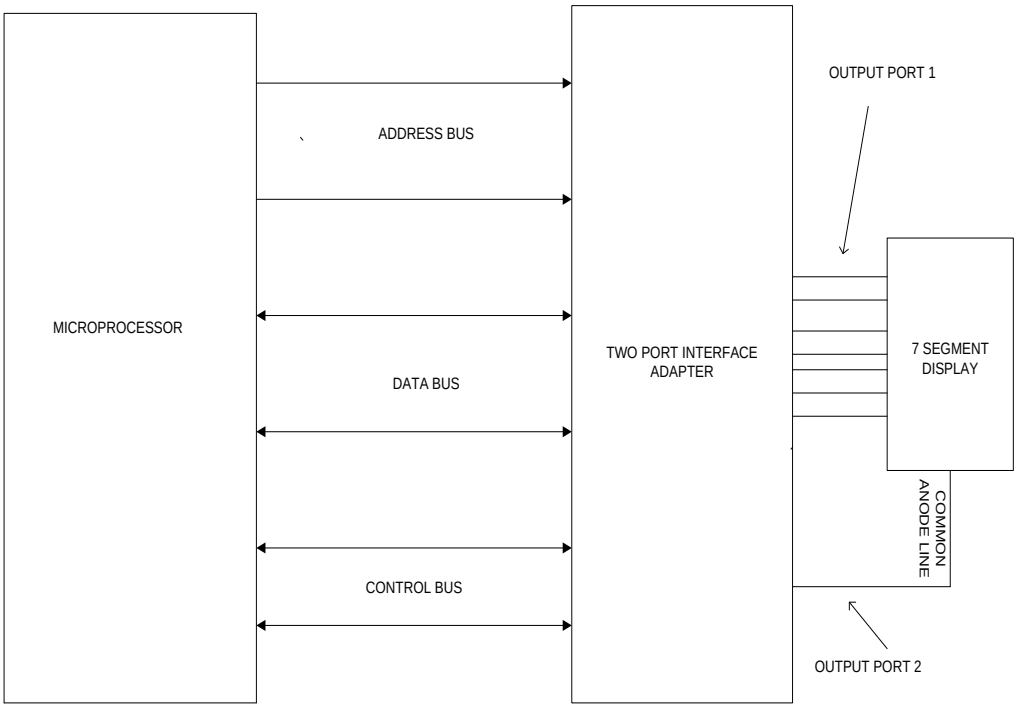
[16 marks]

Q4 (a) In serial transmission data is sent one bit at a time. In order for the receiver to accept data synchronization between the receiver and transmitter is necessary. This is done with synch pulses in the case of synchronous transmission systems and with start and stop framing bits in the case of asynchronous systems. Both receiver and transmitter must use the same format: that is the speed of transmission, the number of bits expected and error checking format if used.

In parallel transmission all data bits are sent in parallel. This method is used over short distances due to the relative complexity and cost of cabling. Even though there are no extra parallel to serial or serial to parallel conversion devices needed and data is available directly from the output ports of the microprocessor problems arise due to the number of data lines needed and the ground return.

[8 marks]

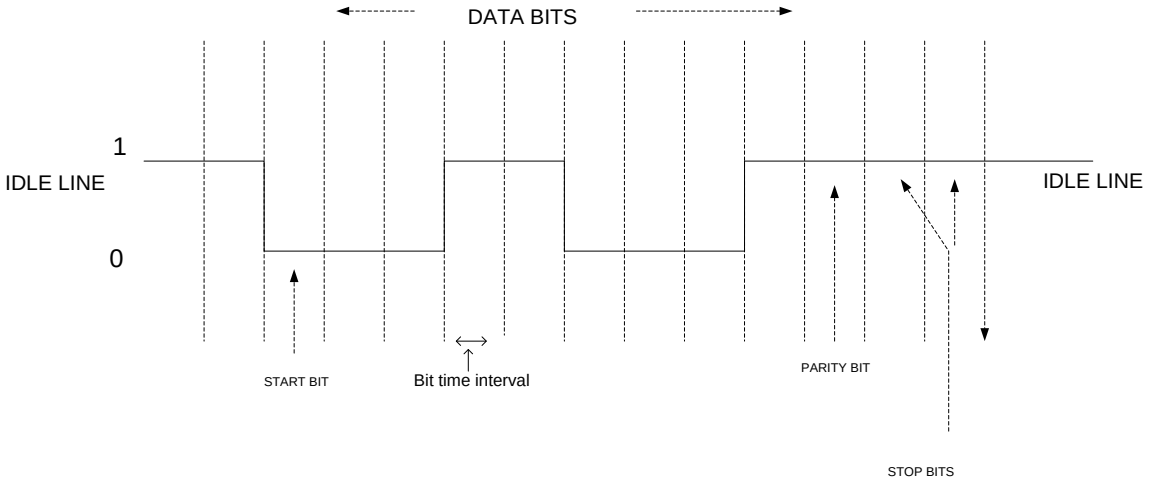
(b) Block diagram of display interfaced to a microprocessor:



[10 marks]

(c) The Baud rate determines the time interval between bit changes. Synchronization is achieved by framing the data with start and stop bits. Error checking is done by counting the number of ones in the data and setting a parity bit to a one or zero depending on whether odd or even parity is used.

Serial waveform



Since the baud rate is 4800 the bit time interval is 1/4800 = .208 mS Since 11 bits are sent the total time is 11.(.208) = 2.288 mS. The maximum transmission rate is thus one byte every 2.288 mS which is 437 bytes per second.

[15 marks]

SEMESTER 1 SUPPLEMENTAL 2011 SOLUTIONS

Q1 (a) The branch instruction uses relative addressing and where it branches to is dependant on where the branch instruction is located in program memory. Since this uses relative addressing the offset is limited to 127 steps forward ($7F) or 128 steps backward ($80).

The jump instruction will always cause program execution to continue at the specified address regardless of where in program memory the instruction is located. The jump instruction uses extended addressing.

Thus program execution will continue from the same location in memory regardless of which instruction is used provided the offset from current location is within the relative addressing limits.

[10 marks]

(d) (i) The 6800 microprocessor has the following registers:

(i) Accumulators A and B (both 8 bit),

(ii) A 16 bit index register,

(iii) A 16 bit stack pointer,

(iv) An 8 bit condition code register and

(v) A 16 bit program counter.

The 6809 microprocessor has in addition to the above:

(i) A second 16 bit index register and

(ii) A user stack pointer and

(iii) An 8 bit direct page register.

(ii) The 6800 micro has 78 instructions and the 6809 has 59. Some instructions were replaced by more general ones which the assembler would translate, and some were even replaced by addressing modes.

While the 6800 had a fast 8 bit mode to address the first 256 bytes of RAM, the 6809 had an 8 bit Direct Page register to locate this fast address page anywhere in the 64K address space.

(iii) The 6800 micro was capable at running at up to 1 MHz while later versions of the 6809 could run at speeds up to 4 MHz.

(iv) Both the 6800 and the 6809 are housed in 40 terminal DIP but they are not pin compatible as the extra interrupt and clock features of the 6809 require additional pin usage.

[13.3 marks]

(c) The maximum offsets available are 127 steps forward and 128 steps backwards. In hex the offset range is $7F - $80.

According to the instruction set BGE tests the V (overflow) bit and the N (negative result) bits and will branch under the following conditions:

If the result is not negative and no overflow has occurred then the result must be ≥ 0 and branching will occur.

If the result is negative and overflow has occurred then the result is ≥ 0 and branching will occur.

Otherwise the result is less than zero and no branching takes place.

CCR H I N Z V C

0 0 0 0 0 0 OR

0 0 1 0 1 0

[10 marks]

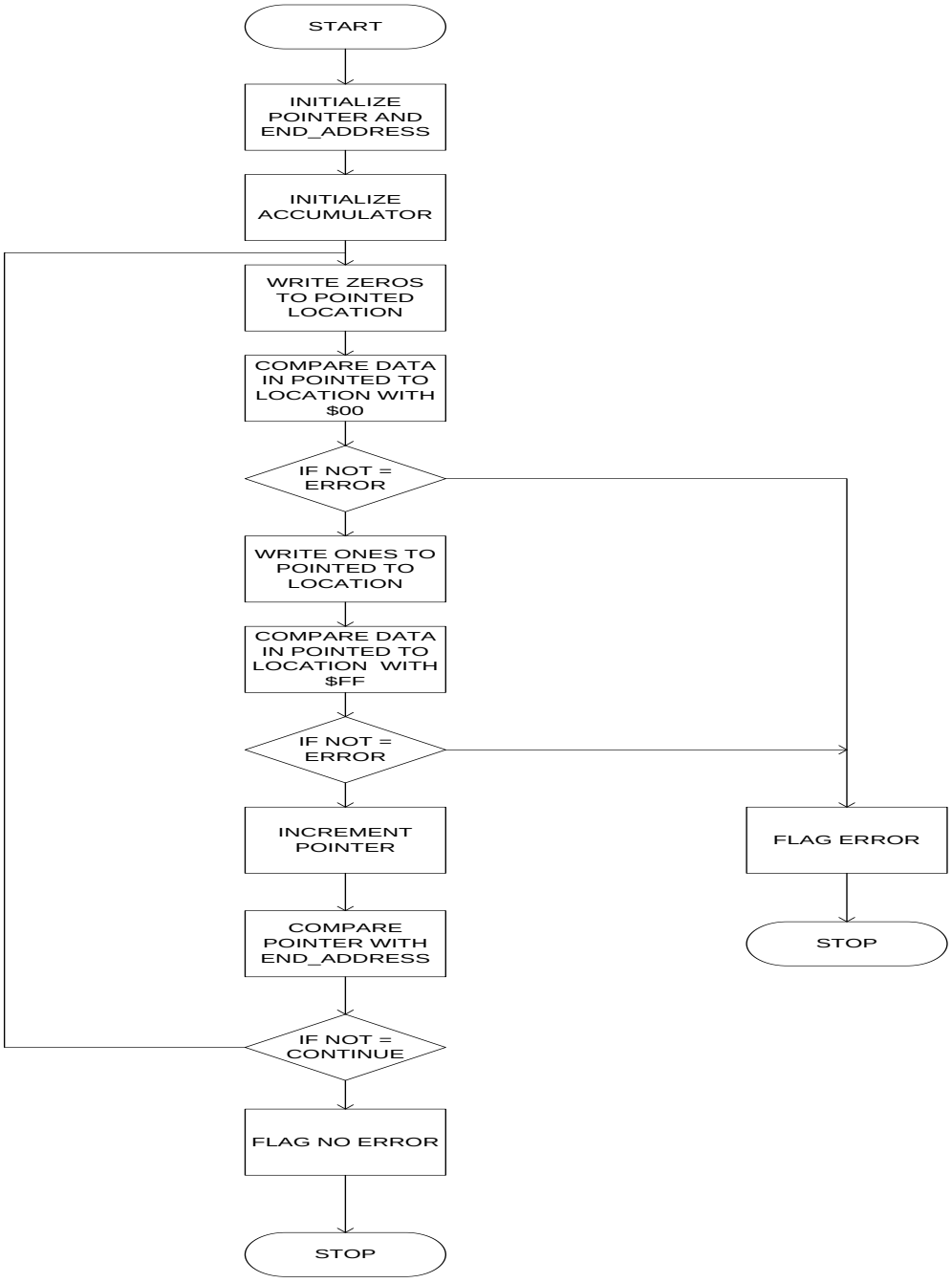
Q2 (a) To test ROM it is necessary to add together the contents of all the ROM locations and ensure that the result is always the same. This result is called the CHECKSUM. If the checksum is different it means a ROM location is faulty. If the checksum is a 32 or 16 bit number then the addition process must take account of carries generated.

To test RAM it is necessary to ensure that all locations can be written to and read from and also that all bits in the location can be changed. To do this it is necessary to first write zeros to the location and then check that zeros can be read followed by writing all ones to the location and checking that these ones can be read.

This procedure must be followed for all RAM locations to be tested.

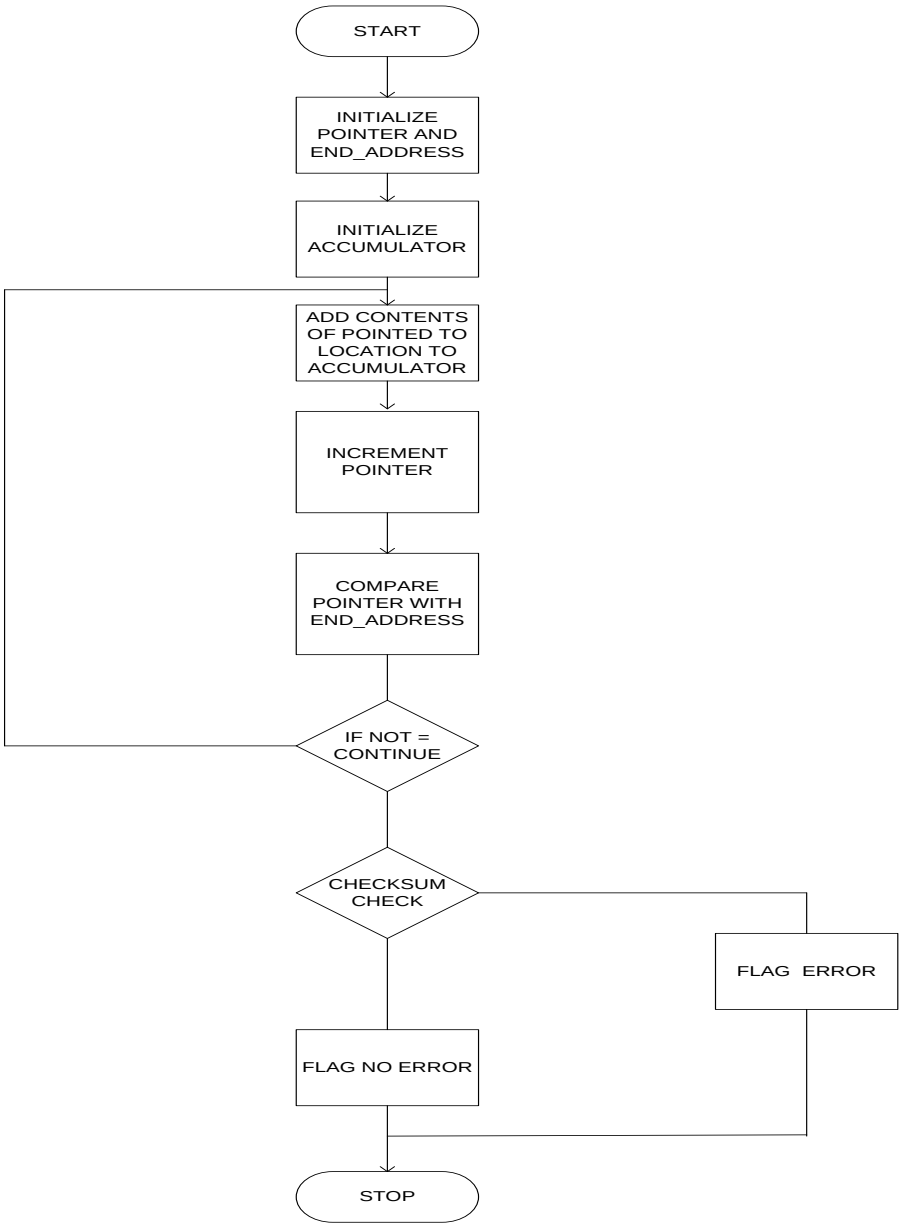
[10.3 marks]

(b) Flowchart for testing RAM.



[11 marks]

(b) Flowchart for testing ROM.



[12 marks]

Q3 (a) The 6800 series of 8 bit microprocessors have 16 address lines and this gives a capacity of addressing up to 16^16 = 65536 locations

[3 marks]

(b) A microprocessor system requires different types and sizes of memory For example it needs ROM for monitor programs, RAM for data storage, EPROM for development and in the case of the 6800 series allocated memory space for memory mapped I/O purposes. Hence there is a need for additional address decoding circuitry to select each of the different memory devices attached to it each with a different address range.

[7 marks]

(c) Logic Symbol for 3 to 8 line decoder:

Truth Table for 3 to 8 line decoder

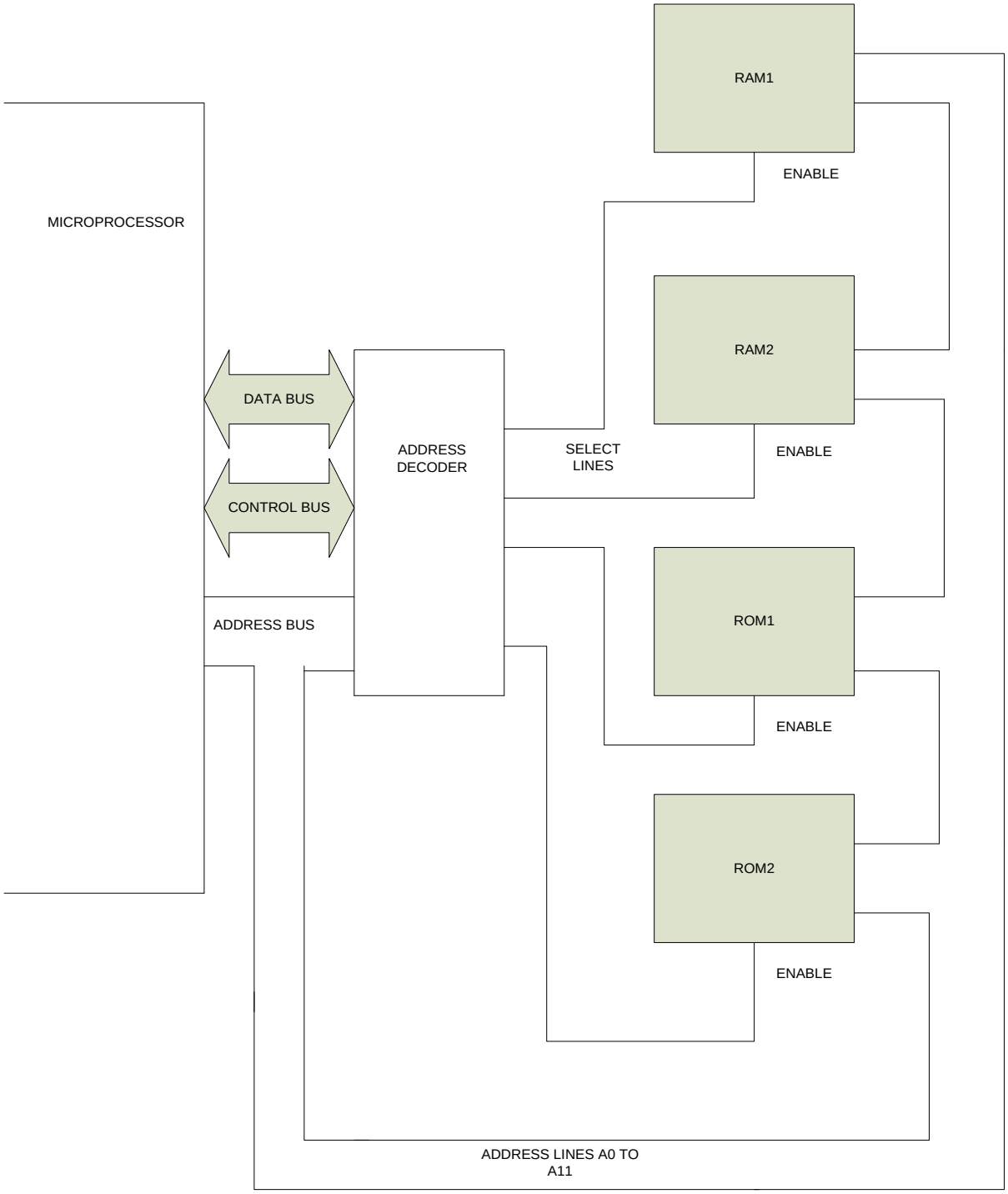
INPUTS OUTPUTS

S2	S1	S0	O7	O6	O5	O4	O3	O2	O1	O0
0	0	0	0	0	0	0	0	0	0	1

0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

[7 marks]

(d) Block diagram for address decoding circuitry.


RAM1
ENABLE
MICROPROCESSOR
RAM2
DATA BUS
ADDRESS
DECODER
SELECT
LINES
ENABLE
CONTROL BUS
ROM1
ADDRESS BUS
ENABLE
ROM2
ENABLE
ADDRESS LINES A0 TO
A11

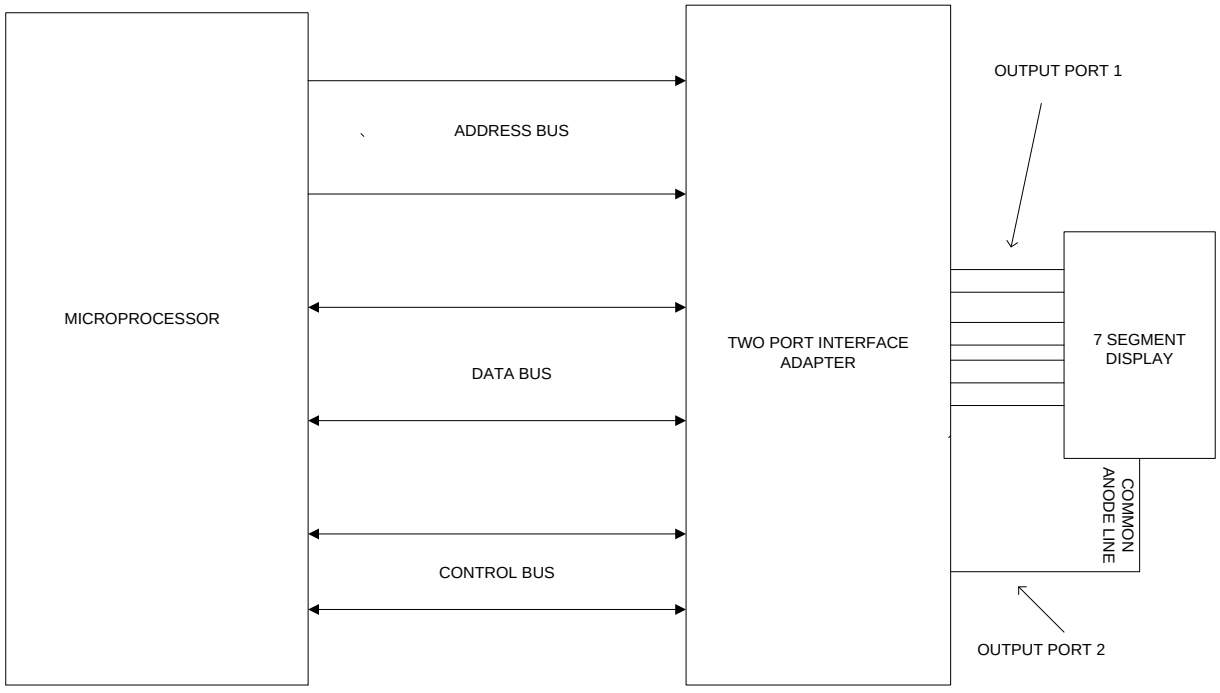
[16.3 marks]

Q4 (a) Since the 6800 family of microprocessors have no direct I/O capability additional devices known as ports are needed to interface I/O devices. These port devices are connected to the microprocessor via the address, data and control buses and are treated by the microprocessor as memory elements in that data to be output is written to a port and data to be input is read from a port (Memory mapped I/O).

Typically these port devices are programmable via internal status or control registers so that a port device can either be an input port or an output port. Hence 7 segment displays and keypads can be interfaced to a 6800 microprocessor using these port devices.

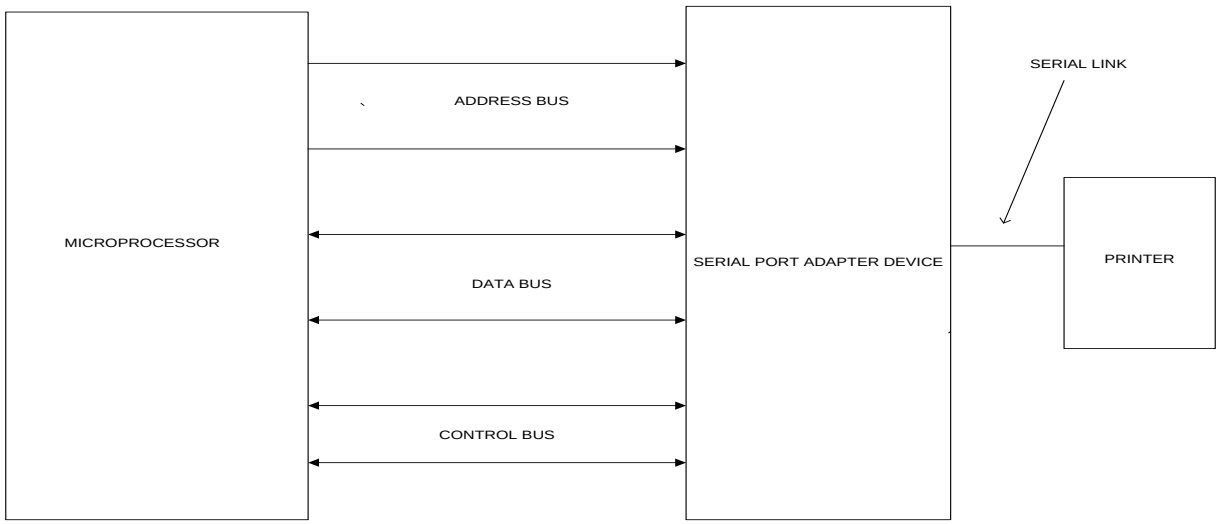
[10.3 marks]

(b) Block diagram of display interfaced to a microprocessor using parallel transmission of data:



[12 marks]

(e) Block diagram of a microprocessor using serial transmission of data:



[11 marks]