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, decode 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 decoding and 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 or working registers.

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

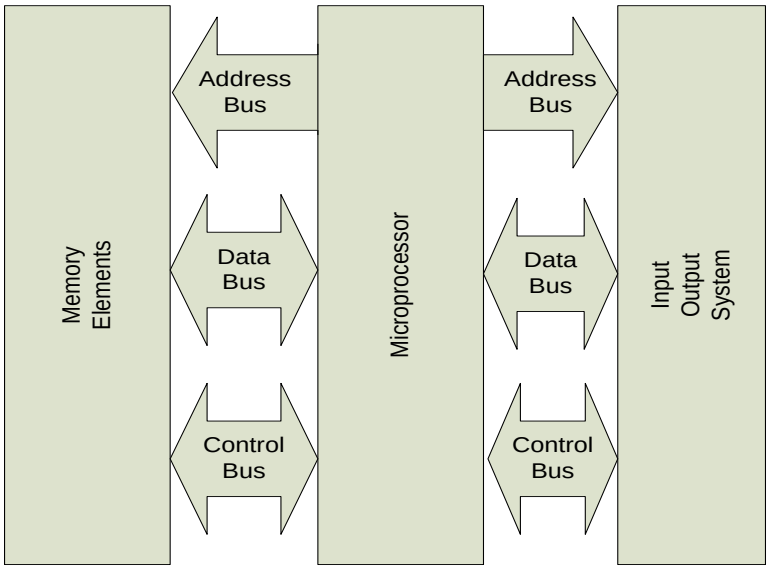


Figure 1

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

Memory, microprocessor and I/O system.

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

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.

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 an early microprocessor, the Motorola 6800 8 bit microprocessor, showing the internal registers and the pins associated with the three buses described previously.

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

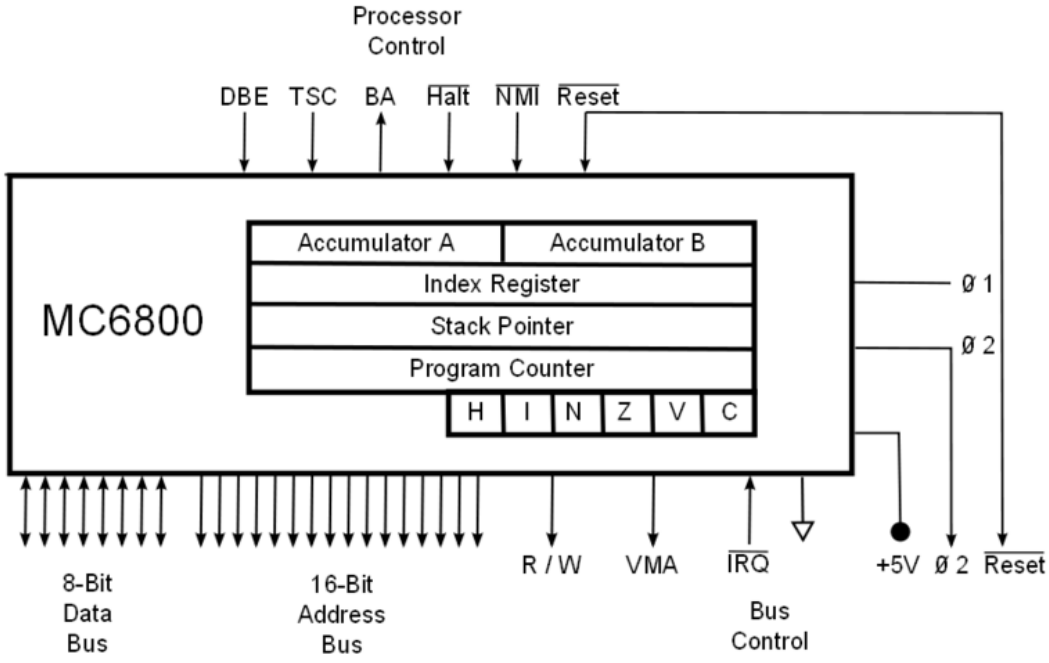


Figure 2

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.

A more modern microcontroller is the Programmable Interface Controller (PIC) and one such PIC is the 16F88 and a pin-out is shown below (Figure 3)

This PIC has all of the elements of Figure 1 above on one integrated circuit (IC) or 'chip' and thus is a self-contained ***microcontroller.***

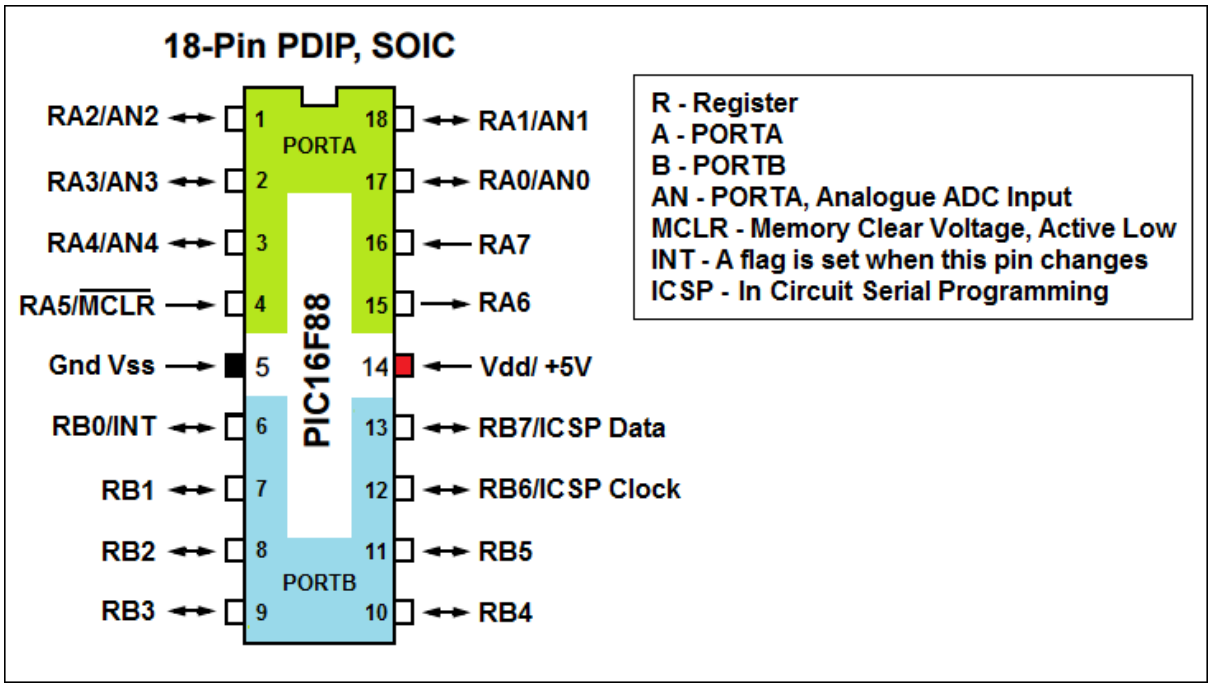


Figure 3

This device has one accumulator called the working register, a status register, on board program memory, on board random access memory and two input/output ports as well as other features. As such it can function as a process controller without the need for additional interfacing circuitry unlike the previously mentioned microprocessor, the 6800.

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 AND TIMING

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.

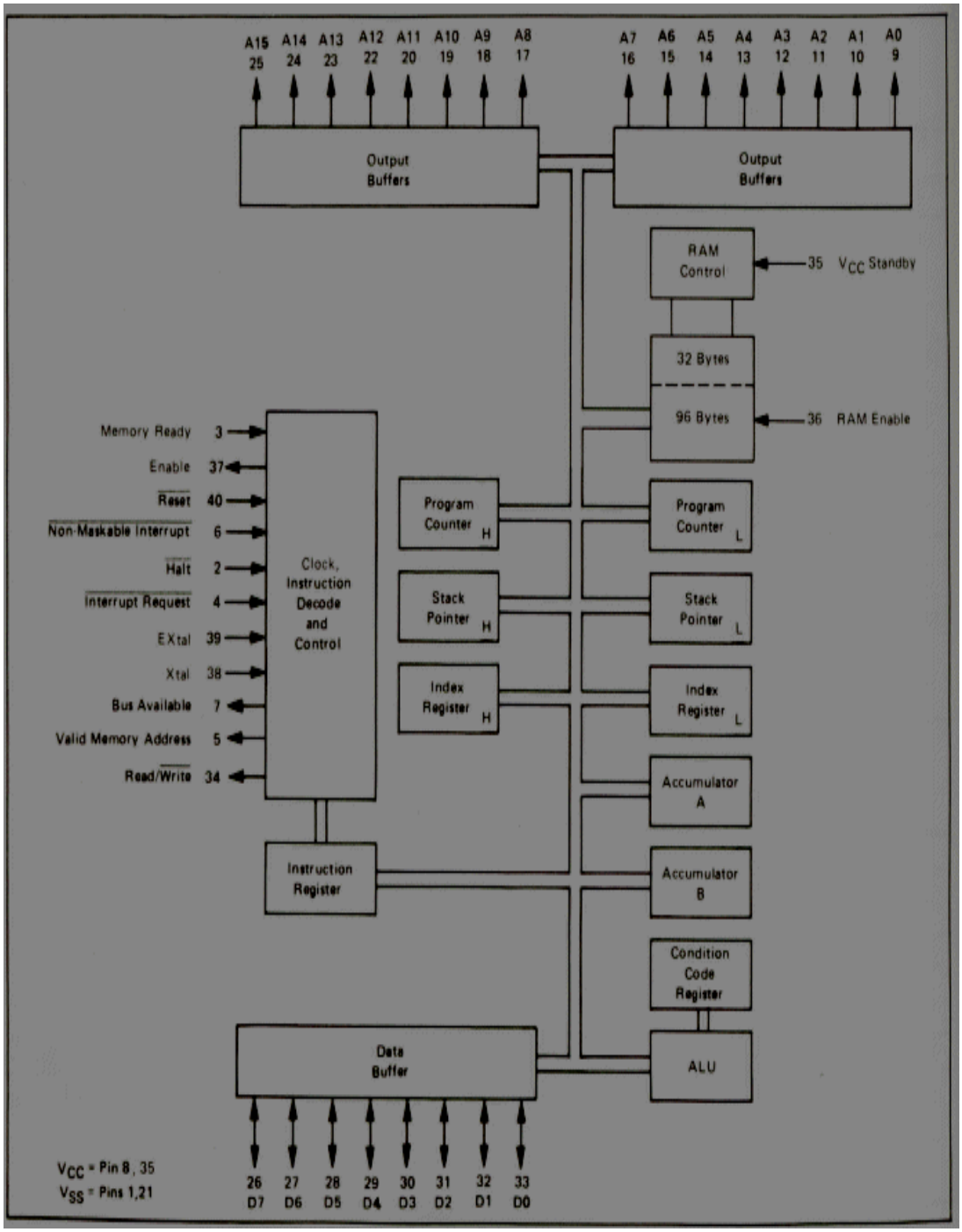
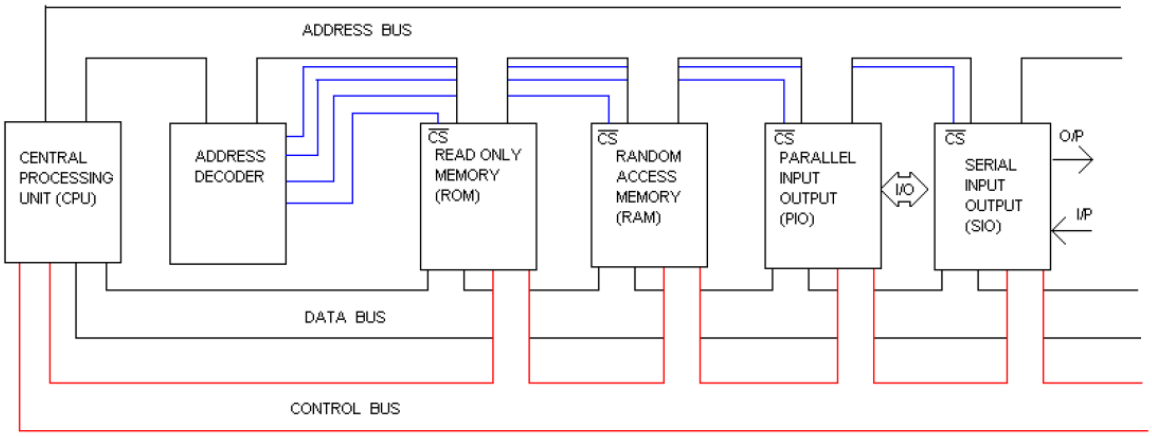


Figure 4

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.

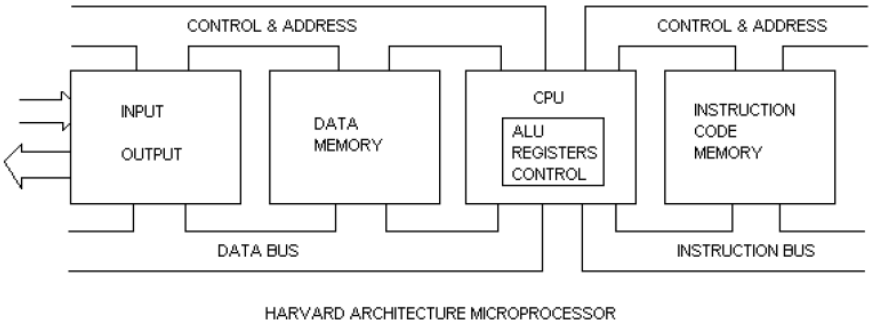
CONTROL BUS - all the other control signals and power for the cpu. Includes power, earth, clock signals, interrupts, controls for the other two buses and connections to other processors.

Comparison of Harvard and Von Neumann Architectures

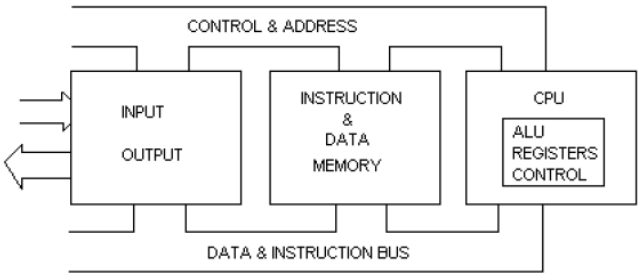
In Harvard architecture, the data bus and address bus are separate. Thus a greater flow of data is possible through the central processing unit, and of course, a greater speed of work. Separating a programme from data memory makes it further possible for instructions not to have to be 8-bit words. For 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

Figure 6

Below is a block diagram of a PIC 16F88 microcontroller. This utilizes the Harvard architecture model and is a RISC microcontroller. It has 4k (14 bits) PROGRAM memory, 368 bytes of RAM, an 8 level (13 bits wide) stack and a 256 byte Electrically Erasable ROM Data memory area all on board with no facility to expand the memory allocation.

Unlike the previous 6800/6809 8 bit microprocessors which have two accumulators, one or two index registers, one or two stack pointers, a condition code register as well as a program counter this PIC microcontroller has only one accumulator called the working register W, a STATUS register and a File Select Register (FSR) for indirect addressing. The PIC does have a program counter and a stack pointer but no instructions to directly access these registers. Since the

number of dedicated internal registers are much less the instruction set is also significantly 'slimmed down' – approx. 70 instructions versus the approx. 250 of the 6809. This PIC (Programmable Interface Controller) also has two 8 bit on board Input/output ports, PORT A and PORT B. Input/output devices such as keypads, light emitting diodes, seven segment displays, liquid crystal displays etc. can be connected directly to the PIC without the need for additional interfacing circuitry.

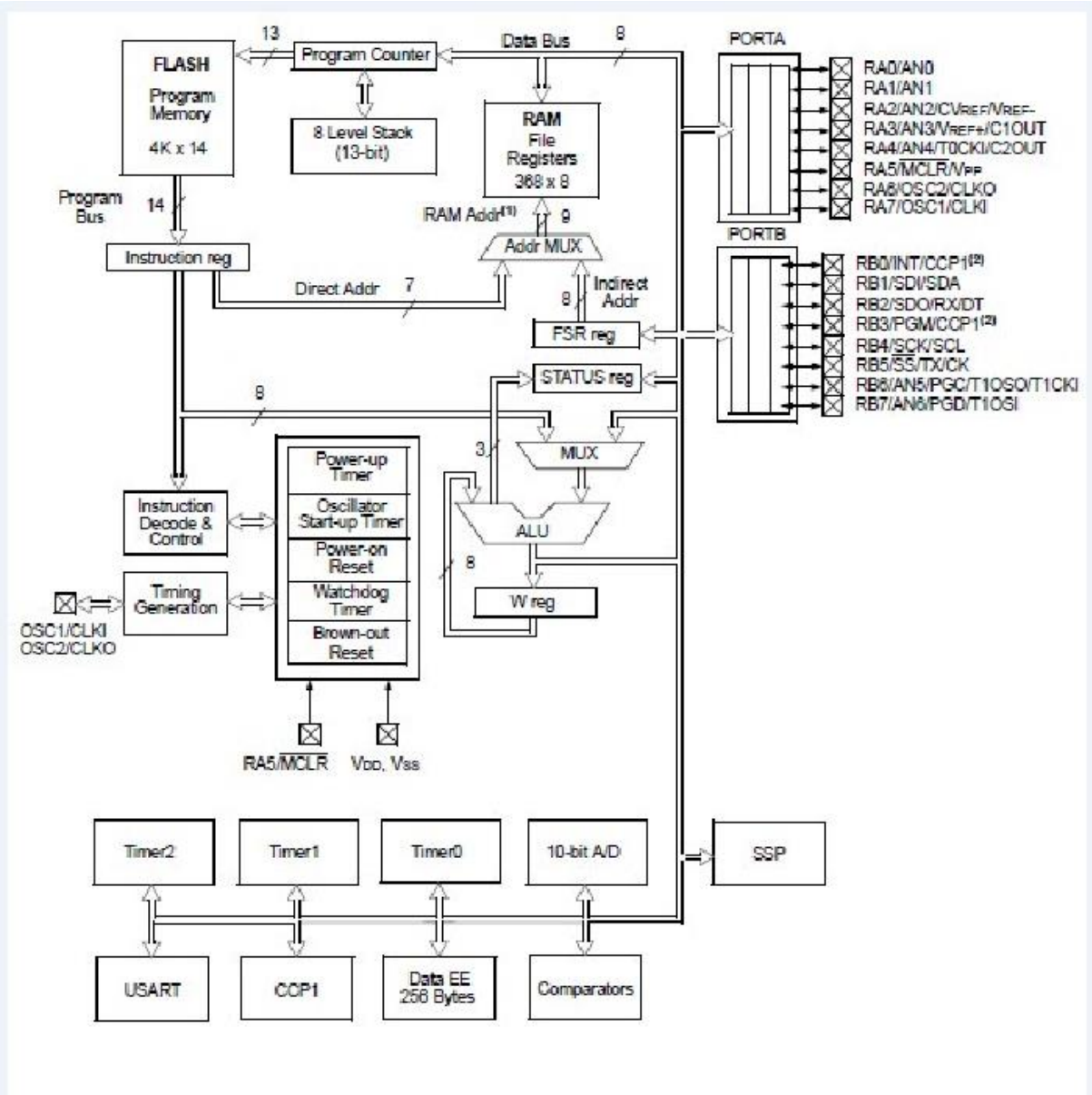


Figure 7

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

SECTION 4

ADDRESSING MODES RELATIVE ADDRESSING AND CONDITIONAL BRANCHING

ADDRESSING MODES IN THE 6800/6809 MICROPROCESSOR

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

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.

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

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.

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.

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

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).

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).

ADDRESSING MODES IN THE PIC 16F88

By comparison the PIC 16F88 has only two addressing modes and they are:

DIRECT

INDIRECT

DIRECT

This is the main addressing mode in the PIC 16F88. For example the instruction

MOVWF PORTB moves or copies the contents of the working register to the memory location called PORTB.

The execution of this instruction is shown below.

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

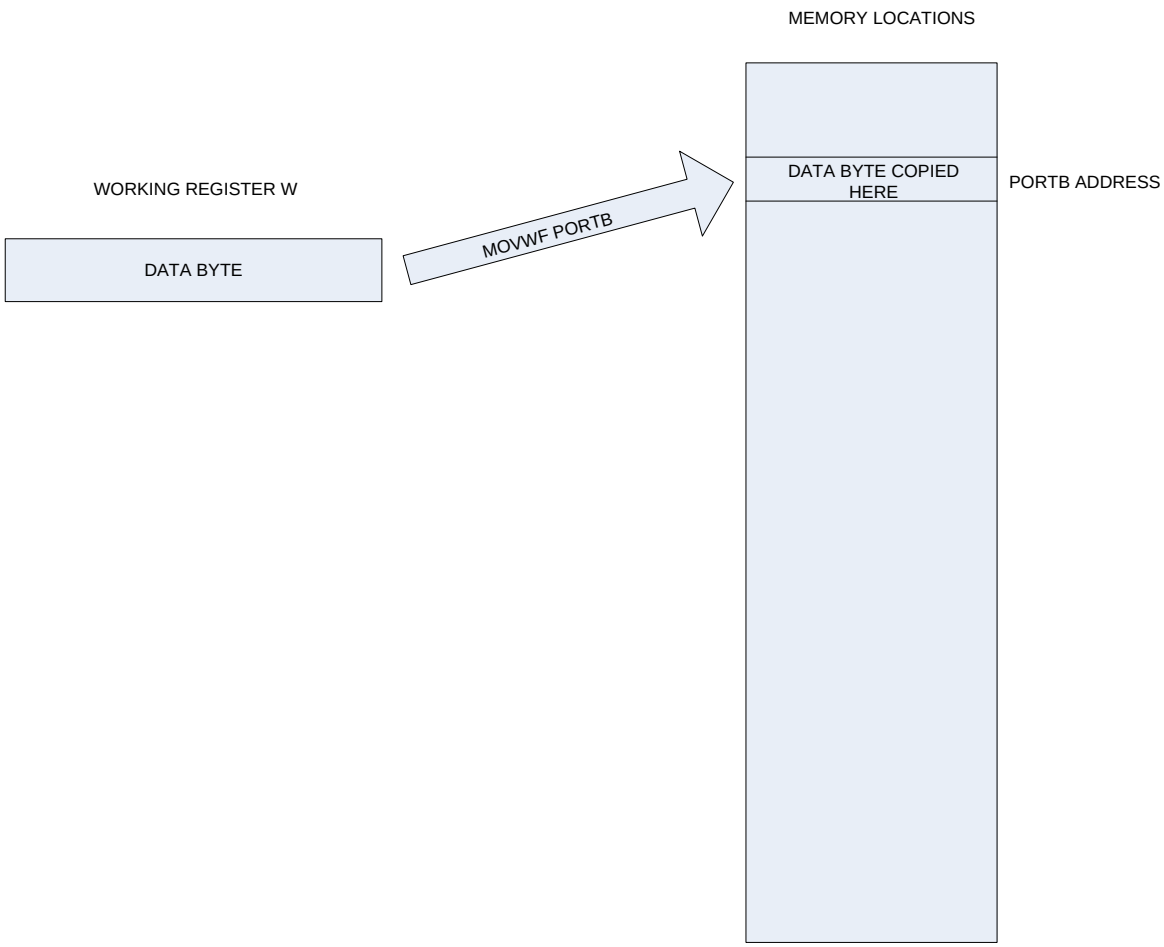


Figure 8

Note that the working register occupies a separate area within the PIC to the main RAM memory area.

Similarly the instruction

MOVF PORTB, W moves or copies the contents of the memory location called PORTB to the working register. This action is shown below.

Note that the DESTINATION as designated by the d in MOVF <File Name>, d is the working register W.

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

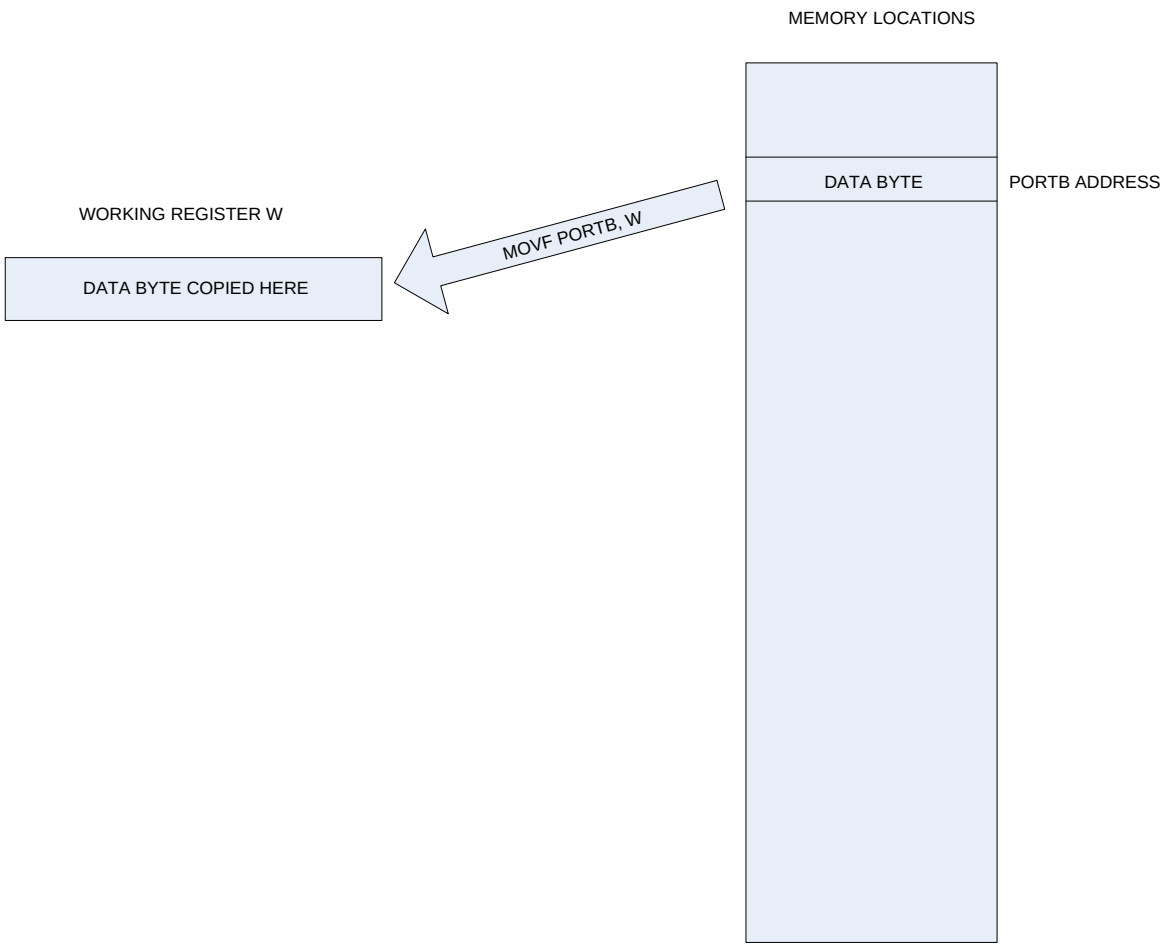


Figure 9

INDIRECT

In this addressing mode the memory location to read from or write to is not specified directly. A FILE SELECT REGISTER (FSR) contains the address of the location to be accessed and an indirect addressing instruction is used.

For example the instruction

MOVWF IND moves or copies the contents of the working register ***to the file (memory location) pointed to by the file select register.***

This action is shown below

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

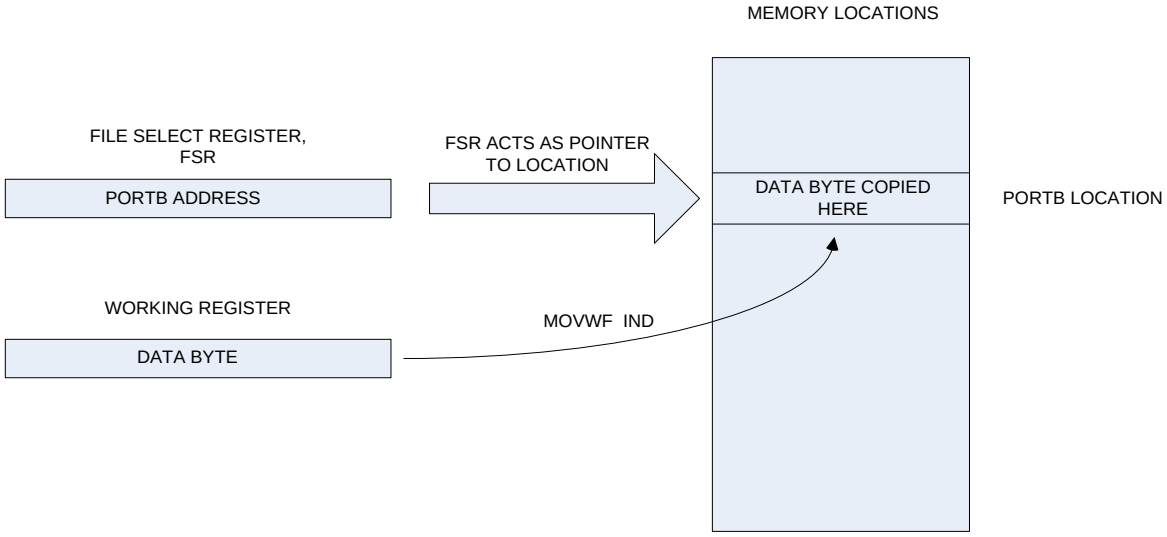


Figure 10

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

Relative Addressing, Conditional Branching and the Status 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'.**

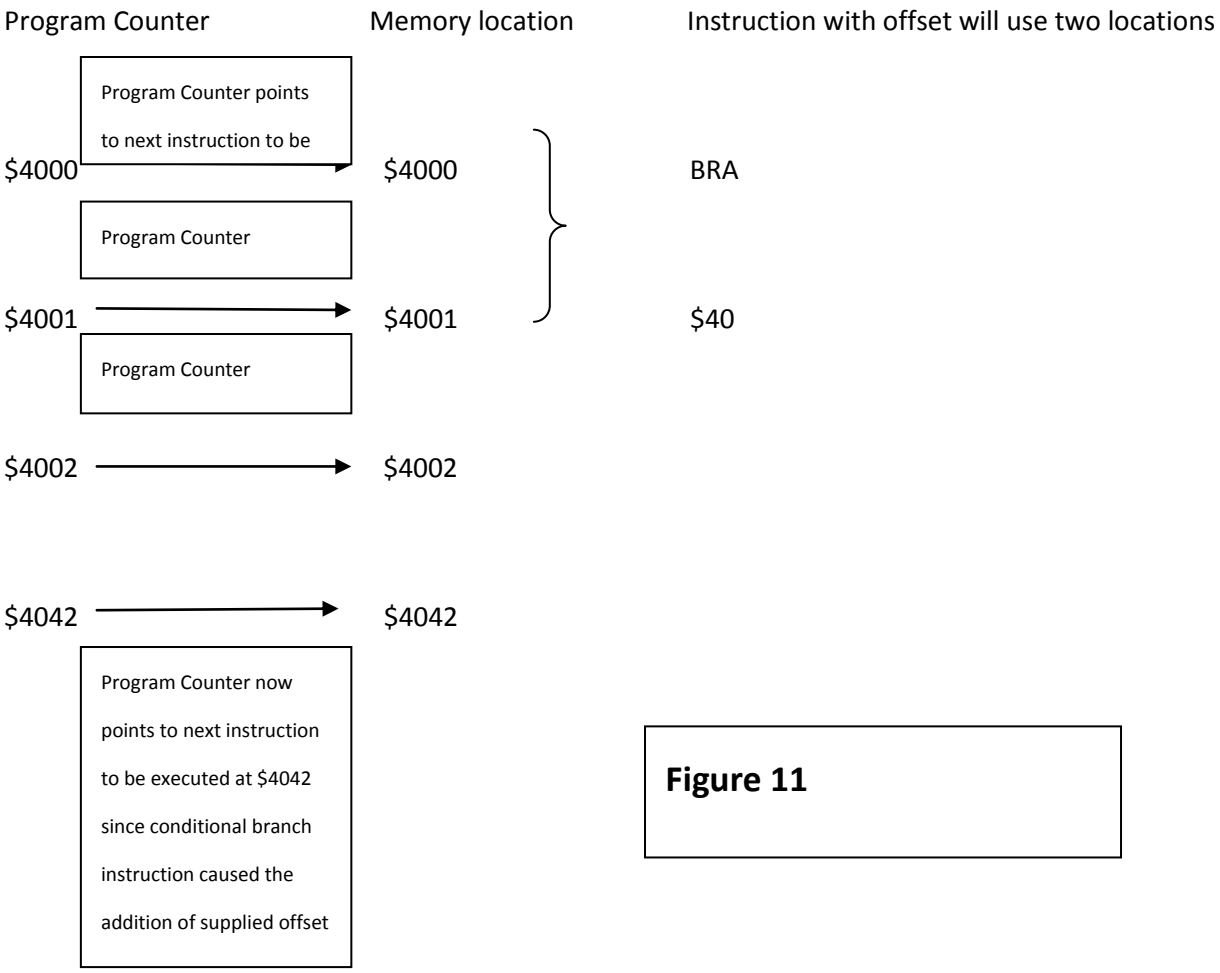


Figure 11

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. The PIC series of micro-controllers have a STATUS register. This is the decision making process in a computer – deciding whether to continue executing the current instructions or to branch or jump to another location to execute different instructions.

For example if the PIC microcontroller meets an instruction such as BTFSS STATUS, Z (which stands for BitTestFileSkipifSet) then the Z or zero flag in the STATUS register is checked to determine whether it is set or not. This flag is set whenever a zero result occurs as a result of an arithmetic or logical operation. Here the file to be tested is STATUS and the bit to be tested is the Z bit. If the Z bit is set then the NEXT instruction is skipped and the following instruction is executed instead.

This instruction that is skipped is usually a GOTO <label> instruction and so the end result of this sequence of instructions is that a decision is taken – do one task if the Z bit is set and do a different task if the Z bit is not set.

For Example in the standard DELAY routine the following instructions occur:

```
        ADDLW   -1
LOOP    BTFSS   STATUS, Z
        GOTO    LOOP
        RETURN
```

Here the arithmetic operation is to subtract 1 from (or decrement) the working register W. If the result is zero then a RETURN (from sub-routine) instruction is executed. If the result is not zero then the GOTO LOOP instruction is executed resulting in the working register being decremented again.

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

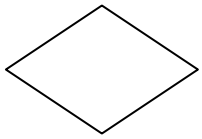
SECTION 5

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.

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.

Note that W refers to the **WORKING REGISTER.** This is an 8 bit register in the PIC 16F88.

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

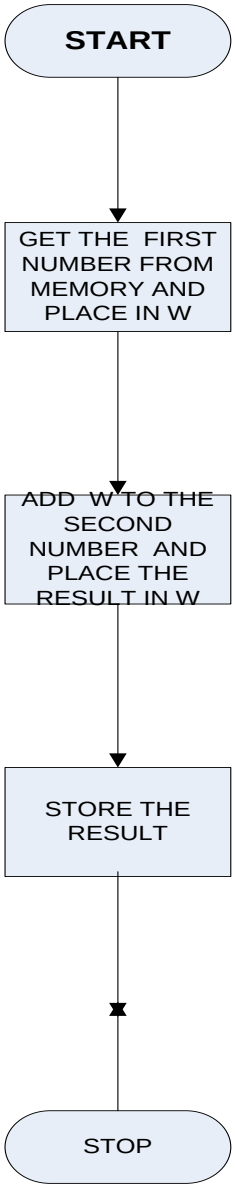


Figure 12

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).

The 16F88 PIC has only one accumulator, called the working register W.

In the 16F88 PIC the assembly language program for the above addition is as follows:

START	MOVF NO_1 , W	This instruction copies the first number from its location to the working register W.
	ADDWF NO_2 , W	This instruction adds the number in the working register to the second number and places the result in the working register W.
	MOVWF RESULT_LOCATION	This instruction copies the result in the working register W to the specified memory location.
	END	This ends the program.

Notes on above:

MOVF moves, or more accurately copies, the CONTENTS of the specified file to the specified destination. Hence the instruction MOVF NO_1 , W copies the contents of the FILE named NO_1 to the working register W.

The file NO_1 is a RAM memory location and this 'link' between the actual memory location and the file name NO_1 is created during the process of writing the assembly language program by

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

using the EQU assembler directive (a command directed to the ASSEMBLER software and not an instruction) as follows:

```
NO_1    EQU    H'20'
```

The above assembler directive specifies that whenever the file name NO_1 is used the location of this file is memory location (hexadecimal) 20.

The instruction ADDWF (File Name), d adds the contents of the working register W to the contents of the file named and the result of this addition can be either placed in W OR placed in the file location. This latter action is specified by the DESTINATION designation d (either W for the working register or F for the file). In our example the destination is to be the working register W and the file name is NO_2 and hence the instruction is

```
ADDWF  NO_2, W
```

As before the location of the file NO_2 is specified by the following ASSEMBLER DIRECTIVE:

```
NO_2    EQU    H'21'
```

Indicating that the file named NO_2 is located in RAM at (hexadecimal) location 21.

The instruction MOVWF RESULT_LOCATION copies the result in the working register W to the file named RESULT_LOCATION which is RAM location hexadecimal 22 as specified by the following ASSEMBLER DIRECTIVE :

```
RESULT_LOCATION  EQU  H'22'
```

The last instruction is END which halts execution of the program.

Hence the full program, including the assembler directives is as follows:

```
            NO_1    EQU    H'20'

            NO_2    EQU    H'21'

 RESULT_LOCATION  EQU  H'22'

START       MOVF     NO_1 , W

            ADDWF   NO_2 , W

            MOVWF  RESULT_LOCATION

            END
```

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

Additional steps which are required when writing a program in assembly language are to specify the origin of the program (where it is to be located in program memory) and what to do or where to go in the event of an interrupt occurring. These steps are shown below.

Comments are inserted after the semi-colon.

```
        ORG 0       ;Reset vector address (when reset either on power on or by pressing ;a
                    reset button) specifies that at memory location 0 is the instruction ;GOTO
                    memory location 5
        GOTO 5      ; Goto start of program i.e. go to location 5.
        ORG 4       ; Interrupt vector address. In the event of an interrupt the
                    ; microcontroller will automatically go to location 4 to find out what ; to
                      do. Here we want it to go to location 5.
        GOTO 5      ; Goto start of program
        ORG 5       ;Start of program memory

          NO_1   EQU   H'20'

          NO_2   EQU   H'21'

RESULT_LOCATION  EQU  H'22'

START     MOVF    NO_1 , W

          ADDWF   NO_2 , W

          MOVWF   RESULT_LOCATION


          END               ; final statement
```

A FLOWCHART WITH DECISION (CONDITIONAL BRANCHING)

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

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

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.

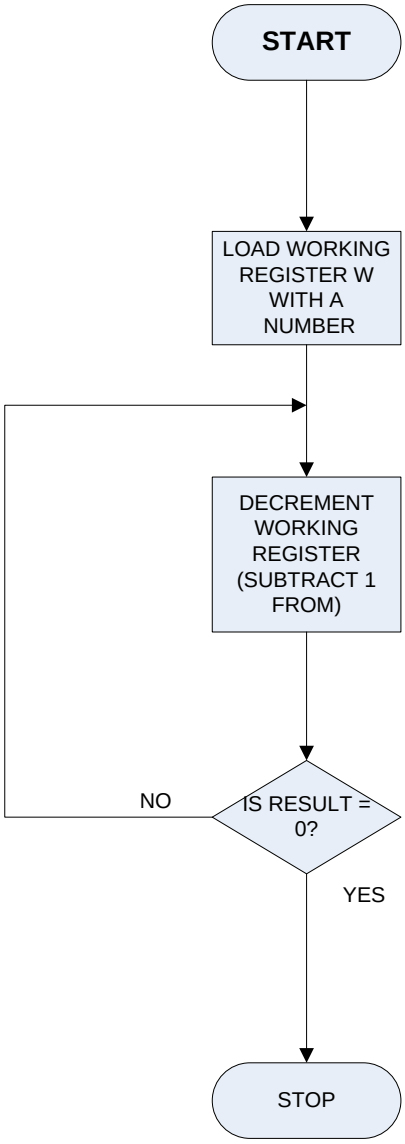


Figure 13

The PIC program for this delay (sub) routine is as follows:

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

```
DELAY   MOVLW   D '255'        This moves a literal value 255 or $FF
                                into W
LOOP    ADDLW   -1              This adds -1 (subtracts 1 from, or
                                decrements, W)
        BTFSS   STATUS, Z       This is the zero check or conditional
                                branch.
        GOTO    LOOP            If the result is not zero go back and
                                repeat.
        RETURN
```

This subroutine program places a decimal number 255 into the working register (since the working register is an 8 bit register this is the largest possible number that can be placed into it) and then 'counts' it down to zero by continually adding -1 to it until the result is zero. This condition (a zero result) is checked by the instruction Bit Test File Skip if Set (BTFSS) on the Z (zero) bit of the STATUS register. If the condition is not met (the result is NOT zero) the instruction GOTO LOOP is executed which causes the program to go back and repeat the previous steps. If the condition is met (the result IS zero) then that instruction is skipped and the next instruction is executed instead. This is the RETURN instruction which causes the program to return to the calling routine. The main program which calls this delay subroutine does so by using the instruction CALL DELAY. The name of the subroutine is DELAY and during the assembly process this name is allocated an address and when called the PIC places this address into the PROGRAM COUNTER.

Also during the assembly process the LOOP label is assigned an address, and when the instruction GOTO LOOP is executed the microcontroller adds a negative number to the current contents of the program counter which results in program execution continuing at LOOP. This is called RELATIVE ADDRESSING and ensures that the DELAY subroutine can be placed anywhere in program memory since the program does not use absolute addresses.

The program does not specify an actual address in memory to loop back to, rather the program 'knows' how many steps it needs to go back from its current position to repeat the previous steps. **This is an example of POSITION INDEPENDENT CODE.**

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

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 25510 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 MOVLW instruction + (time taken for (ADDLW and BTFSS))*255 + time taken for RETURN.

In the PIC each instruction takes one INSTRUCTION CYCLE (which is 4 clock cycles) with the exception of the GOTO instruction (which takes 2 instruction cycles)

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.

Hence the INSTRUCTION CYCLE is 4 µs.

Then knowing the INSTRUCTION CYCLE time we can work out the total delay time:

Total no of INSTRUCTION CYCLES = 1 + (1+1 +2)*255 + 1 ⇨ 1022 ⇨ 4.088ms

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

SECTION 6

SUB-ROUTINES and INTERRUPTS

SUB-ROUTINES

A sub-routine is a separate routine called by the main routine to perform a specific task. The instruction CALL <name of sub-routine> is used to specify which sub-routine is to be called.

For example CALL DELAY or CALL TABLE, calls respectively the delay sub-routine called DELAY or the look-up table sub-routine called TABLE.

When a CALL instruction is met the microprocessor takes the following actions:

- Stores the RETURN ADDRESS in the STACK and
- Overwrites the PROGRAM COUNTER with the ADDRESS of the sub-routine so that the next instruction to be executed is the start of the sub-routine.

Remember that the PROGRAM COUNTER always points to the address of the next instruction to be executed. The PROGRAM COUNTER register can be modified either by a GOTO instruction or a CALL instruction and it can be modified by directly overwriting with a new address or by adding/subtracting to/from it.

The STACK is a special area of RAM memory used both for the temporary storage of return addresses and register values during program execution. In the 6800/6809 microprocessors the stack was a reserved area of (external) RAM and the size of and address of the stack was under user control. These microprocessors also had one or two STACK POINTER registers pointing to the next stack memory location. The stack could be as large or small as desired.

In the PIC series of micro-controllers the STACK is a special area of RAM separate from data RAM and is limited in size.

The stack works on a last-in first-out basis. The last piece of data placed in the stack will be the first piece of data retrieved from the stack. The name comes from a stack of plates or dishes. The last plate placed on the stack will be the first plate to be removed from the stack.

When a CALL instruction is met the micro-controller places the current contents of the PROGRAM counter in the stack and then executes the sub-routines instructions. The last instruction in a sub-routine is always the RETURN instruction and when this is met the micro-controller takes the following actions:

- Fetches the RETURN address from the stack and

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

- Overwrites the PROGRAM COUNTER with this return address so that program execution continues at the next instruction in the main routine after the CALL instruction.

A sub-routine can call another sub-routine which in turn can call another sub-routine etc.

In each case the return address is stored in the stack. The number of sub-routines that can be called in this way is limited by the size of the stack.

In some cases it is necessary to save the current contents of both the working register W and the status register STATUS since the sub-routine execution will use/affect these registers. These register contents can also be saved to the stack and restored from the stack. These actions are done in the sub-routine so that when program control is returned to the main routine all register values are exactly the same as they were before the sub-routine was called.

In some cases the subroutine is called specifically to take action on a particular VARIABLE. For example the DELAY sub-routine might not be a fixed delay sub-routine, rather it may be required that the delay time depends on other factors. In this case the length of the delay can be varied by what is known as PARAMETER PASSING. Here the number to be placed in a register to be counted down to zero (to create the delay) is passed from the calling routine to the sub-routine usually in the working register W.

INTERRUPTS

An interrupt is a 'forced branch' during program execution to another (special) routine, known as an ISR (interrupt service routine). It is caused by a signal from a peripheral and can occur at any time. (Hence the term hardware interrupt – since some piece of hardware generates the signal). Although a timer may interrupt regularly, it is still at 'any time' as far as the program is concerned.

INTERRUPT STRUCTURE

- A special file known as the interrupt control register is used to manage the interrupt mechanism.
- It has a 'GLOBAL INTERRUPT ENABLE BIT' that must be set to 1 for any interrupt to be allowed or occur.
- Each individual interrupt source has its own ENABLE BIT to turn it on/off.
- There is also a FLAG bit for each source of interrupt that is SET by the interrupting signal. This is done regardless of the state of the enables and can be 'polled' to determine which signal has been generated from which interrupt source.

- When an actual interrupt of CPU occurs it then fetches the start address of the ISR from a VECTOR ADRESSING LOCATION (address H'04').
- This routine then runs and the interrupt is 'serviced'.
- The STACK is used in the same way as for subroutines i.e. to store the return address.

Note that if more than one source of interrupt is possible in a program, then the ISR must poll the various flags to determine which one caused the interrupt. This can be done in a particular sequence if there is a need to prioritise the response to different sources of interrupt. More sophisticated systems have a 'built in' priority structure (basically if a 'higher' priority interrupt occurs then it can interrupt a lower priority one).

Interrupt Sequence

Most processors have similar response to an interrupt but the details vary. The main difference between basic devices such as the 16F series and other more advanced systems is context saving (and possibly automatic assignment of priority). This means that some registers (e.g. W and status) are automatically saved in some part of memory (possibly referred to as stack memory). This does not occur in the 16F series; to save the context the programmer must write lines of code into the ISR which simply save the registers to some files (variables). Thus the main sequence of actions we associate with the interrupt process is as follows:

- Finish the current instruction.
- Clear the Global enable flag (preventing further interrupts) and save the current address to STACK.
- Get interrupt vector (load address h'04' where a GOTO start of ISR instruction is located).
- Enter interrupt routine (and if necessary save register context).
- If more than one possible source of interrupt then 'poll flags' to determine source and clear flag to prevent re-entering interrupt when finished.
- Do the task which may be very limited (e.g. counting).
- Restore context.
- RETFIE instruction 'pulls back' return address from stack' and resets the global enable flag.

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

SECTION 7

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:

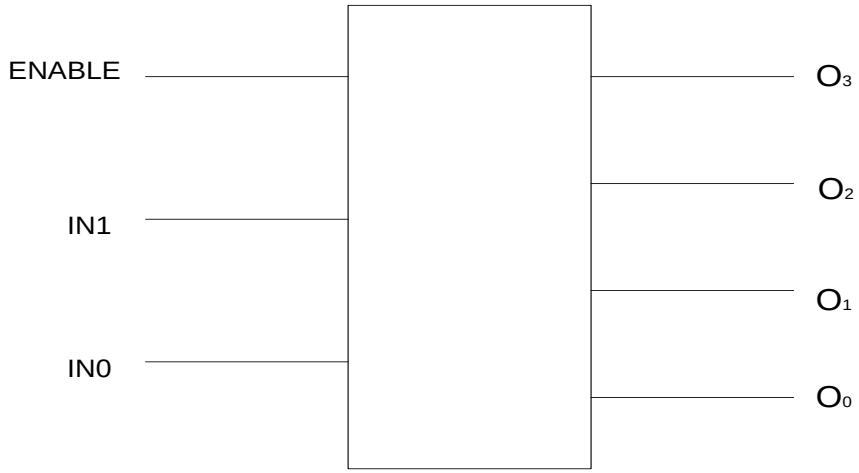


Figure 14

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.

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

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

Figure 15

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 16

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

Figure 17

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.

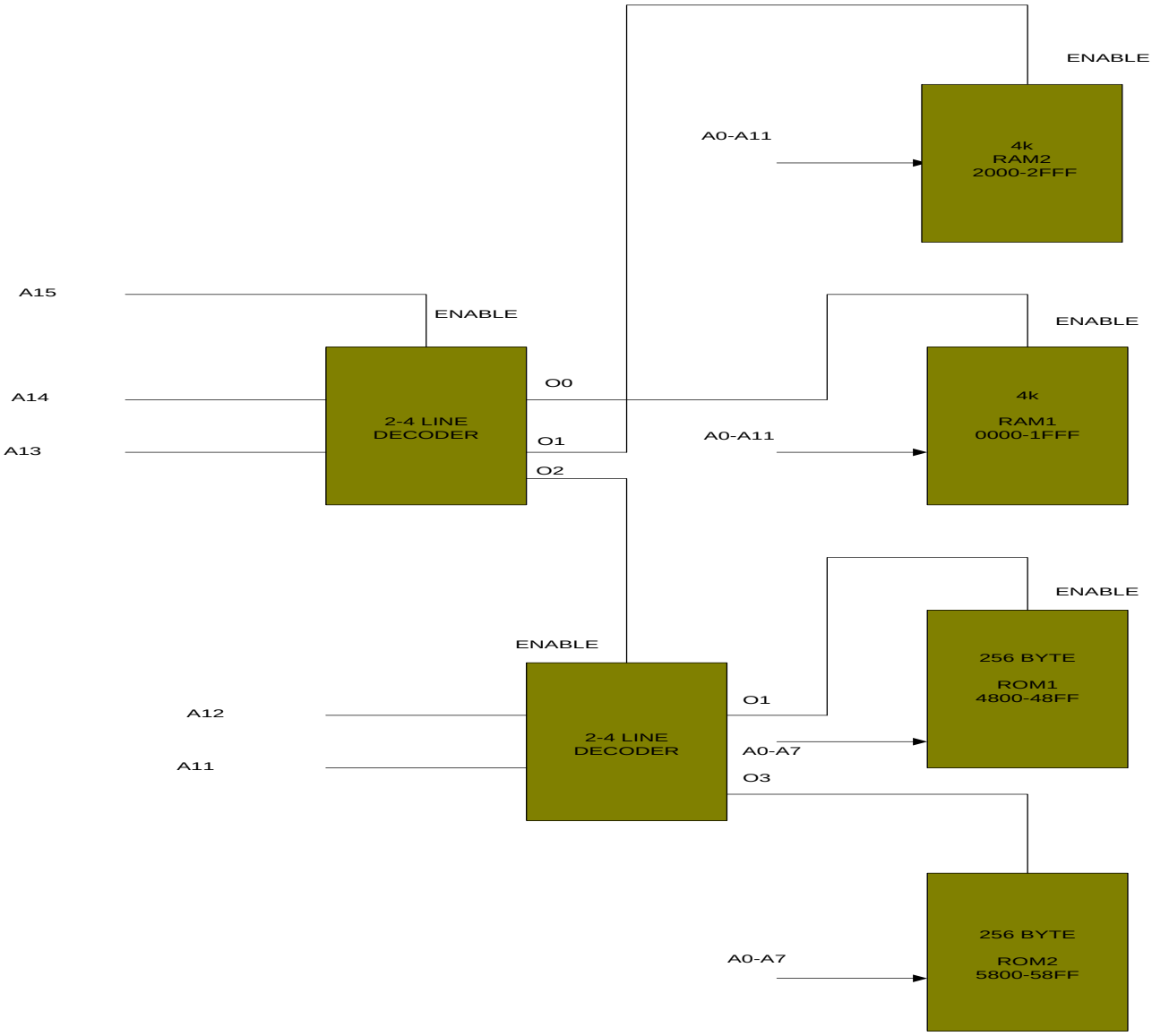


Figure 18

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.

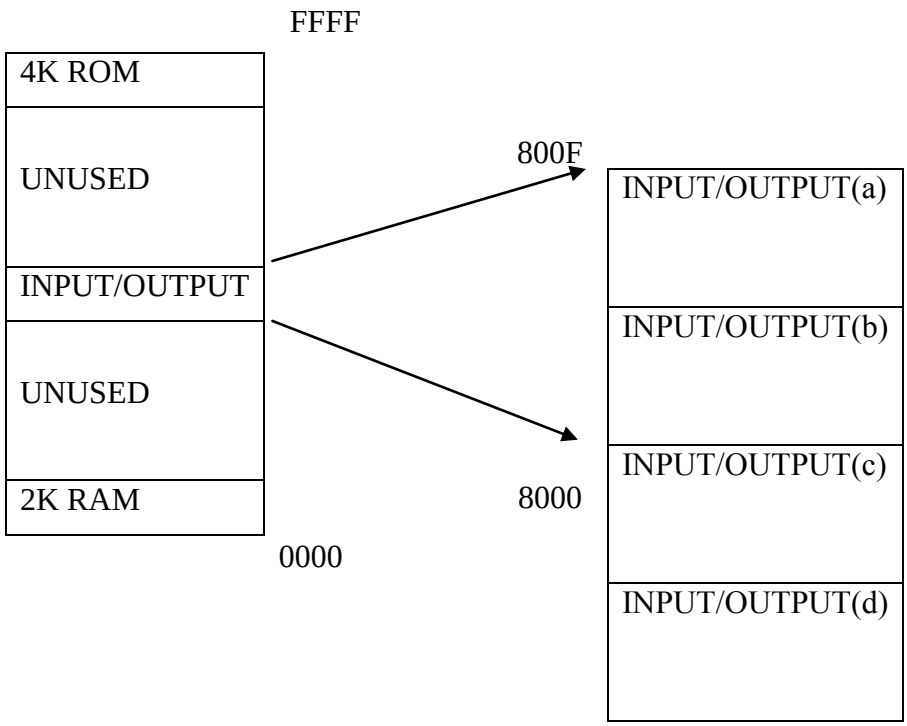


Figure 19

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

Figure 20

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

Figure 21

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

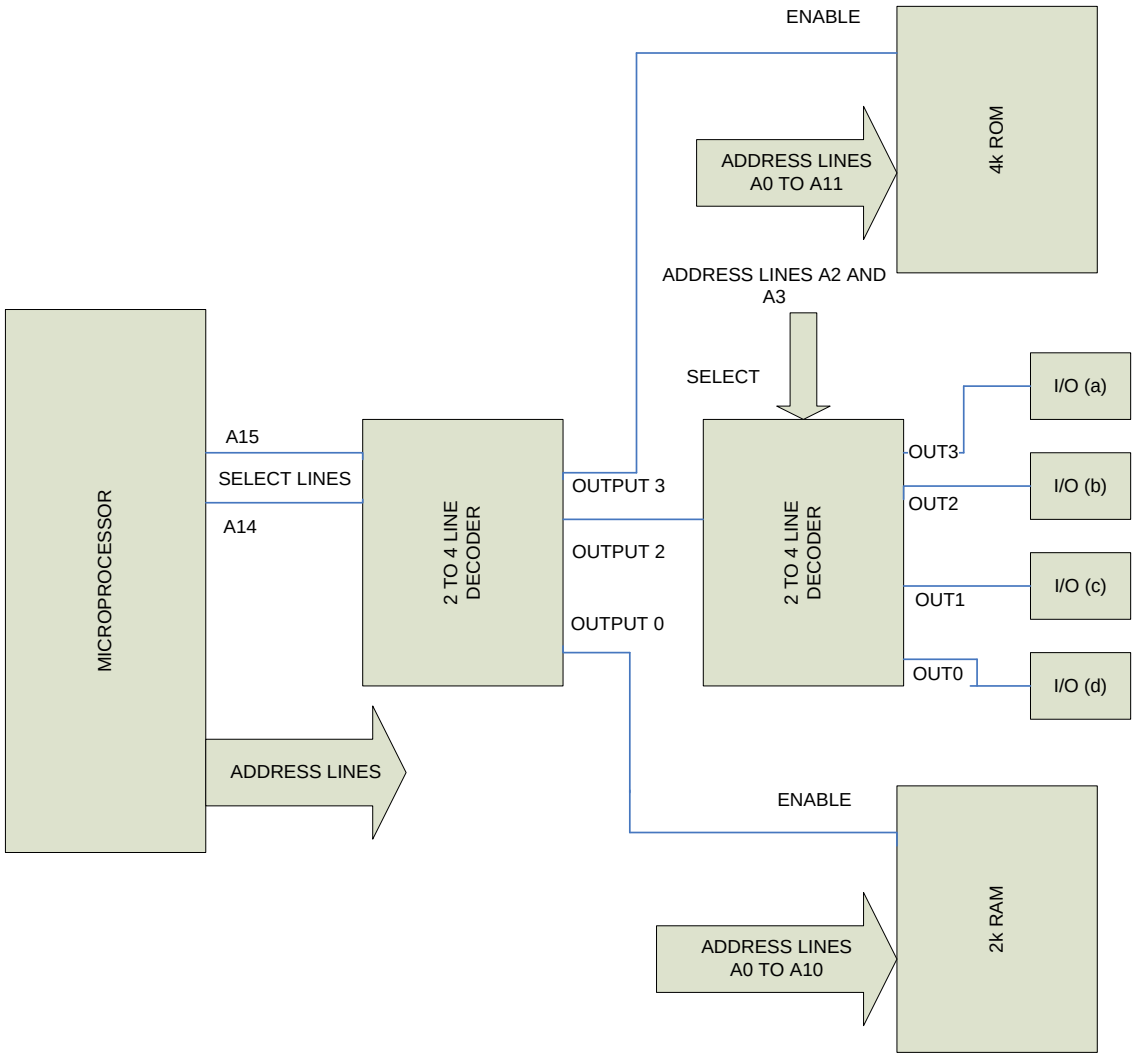


Figure 22

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

Adding three more four port I/O devices

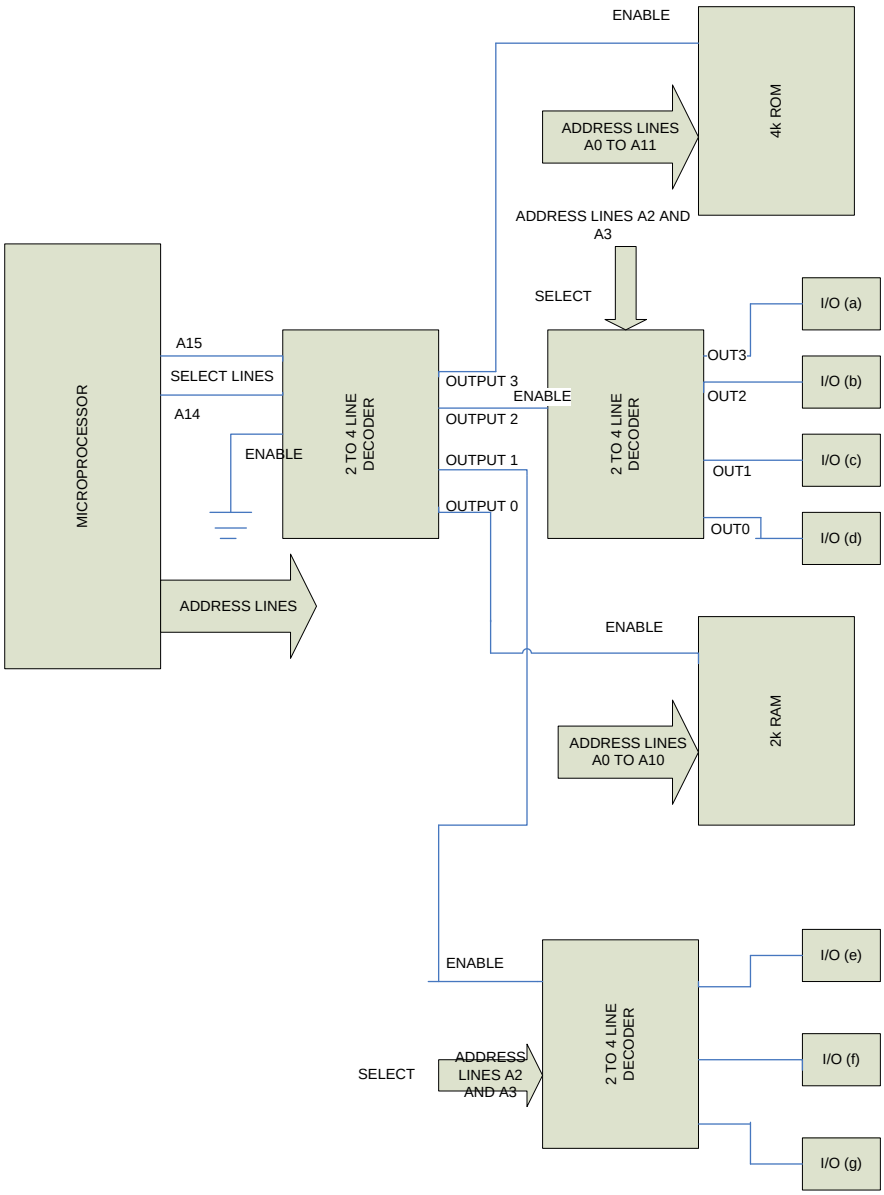


Figure 22

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

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

Figure 23

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

SECTION 8

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).

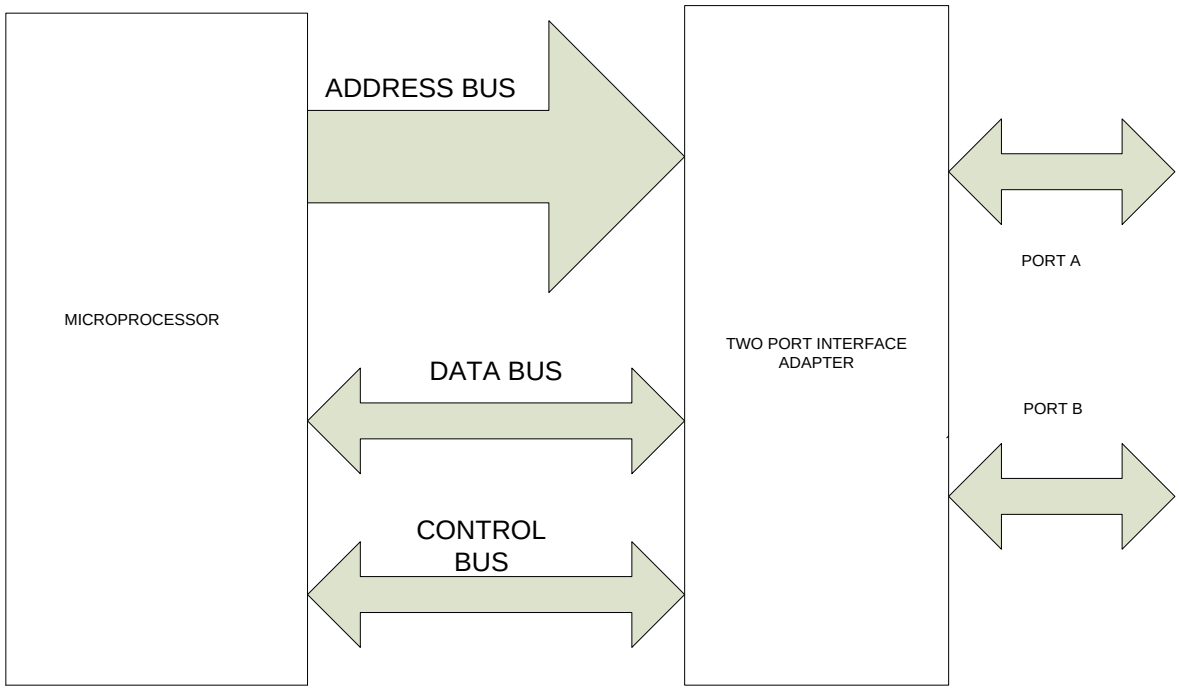


Figure 24

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. In a PIC the corresponding instructions are MOVF PORT_A, W and MOVWF PORT_A.

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

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.

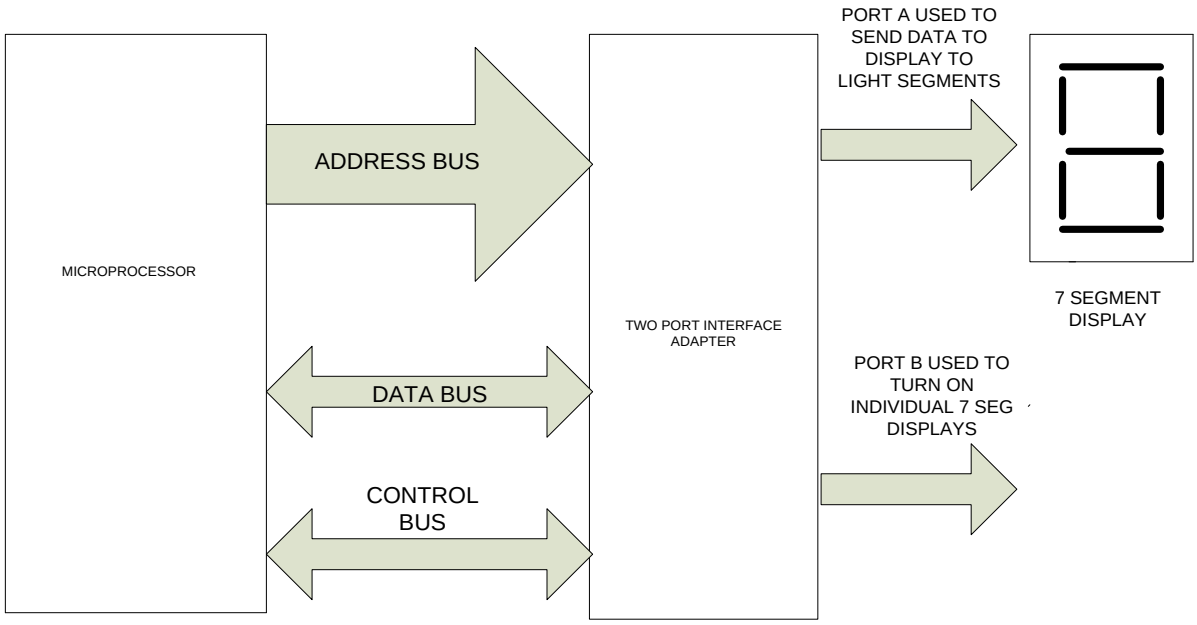


Figure 25

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

The diagram below shows the same 7 segment display connected to PORT A of a typical PIC microcontroller such as the PIC 16F88.

Note that the PIC has dedicated, on-board, I/O ports and hence needs no additional interface circuitry. The PIC also uses I/O memory mapping.

PIC 16F88 Driving a 7 segment display.

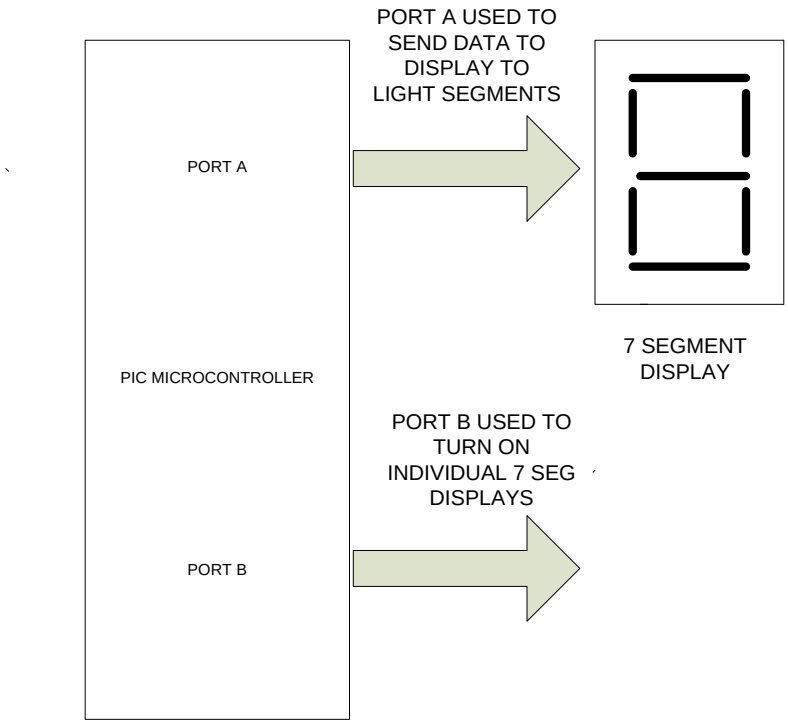


Figure 26

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

As with the previous example to display a decimal digit the correct **SEGMENT DATA** must be output on **PORT A.** In a typical system using **MULTIPLEXED DISPLAYS** the data output on the other port **PORT B** determines which of the displays is turned **ON** at any one time.

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.

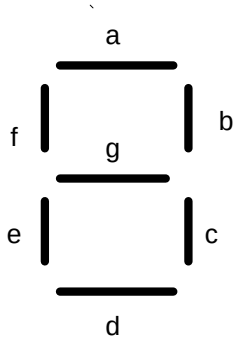


Figure 27

Thus it is possible, by lighting the relevant segments, to display a range of decimal and hexadecimal numbers and alphanumeric characters. For example lighting segments a,b,c,d,e and f will display a 0 (zero). Lighting all segments will display an eight (8).

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

DISPLAY MULTIPLEXING

As an example consider the case of interfacing a number of seven segment displays to a 6800/6809 microprocessor or a PIC 16F88. This is known as display multiplexing:

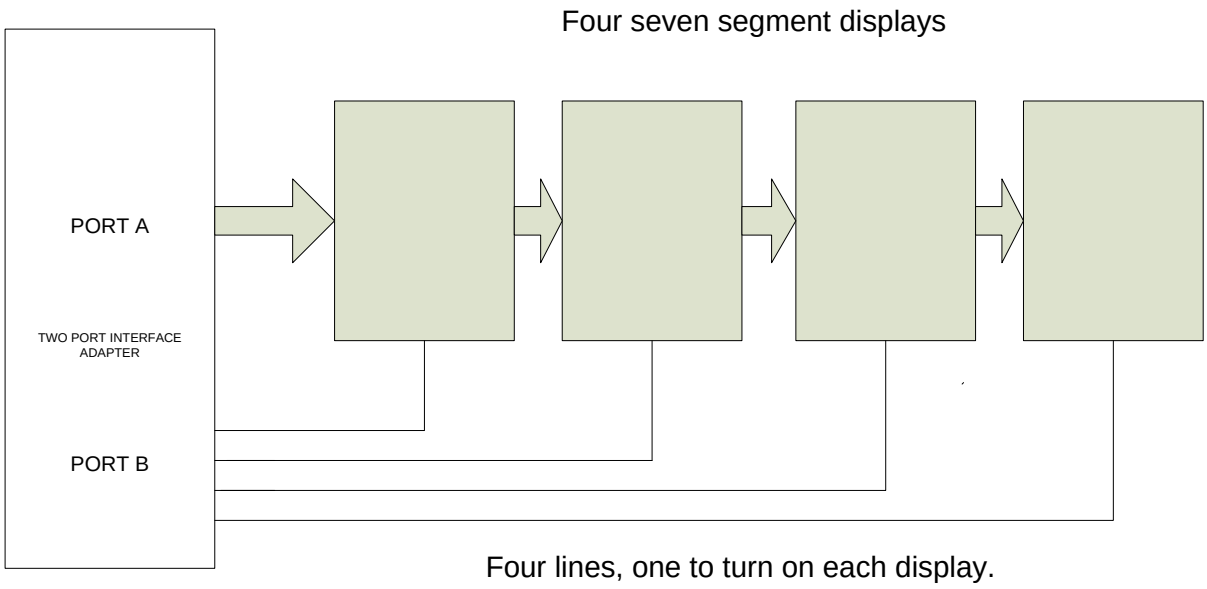


Figure 28

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.

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

As an example consider the case of interfacing a simple keypad to a 6800/6809 microprocessor or a PIC 16F88 microcontroller.

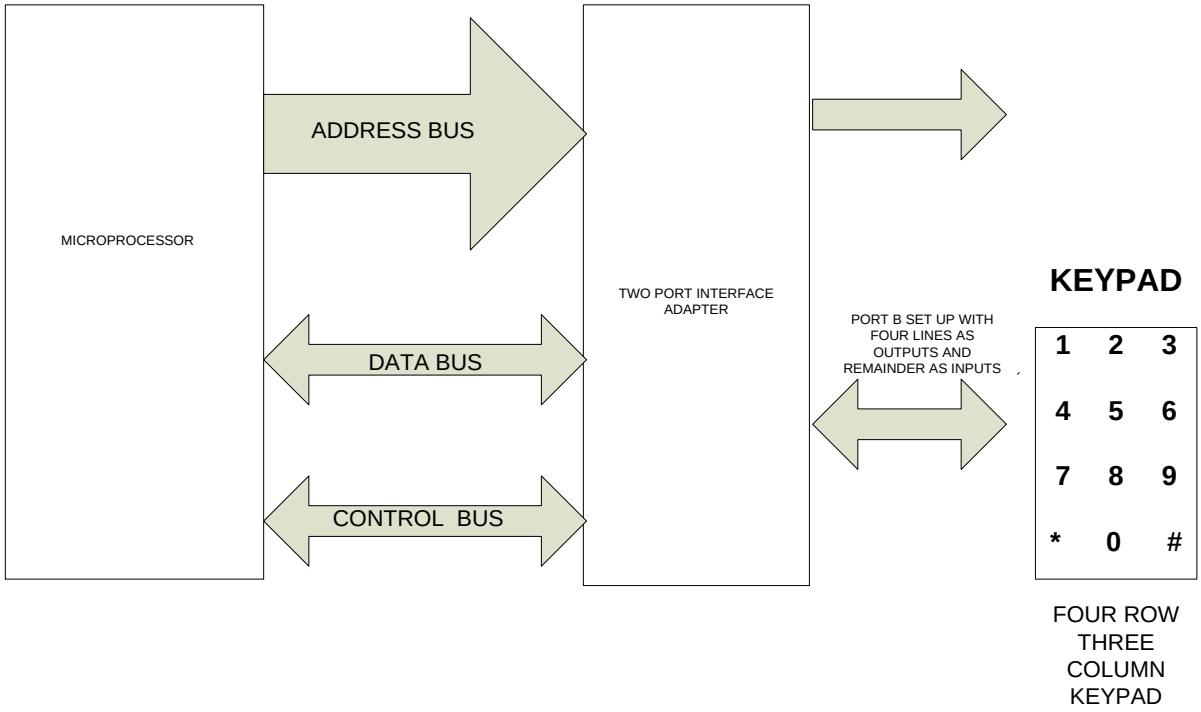


Figure 29

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 de-bouncing 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.

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

SECTION 9

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.

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

Format of Asynchronous Serial Data

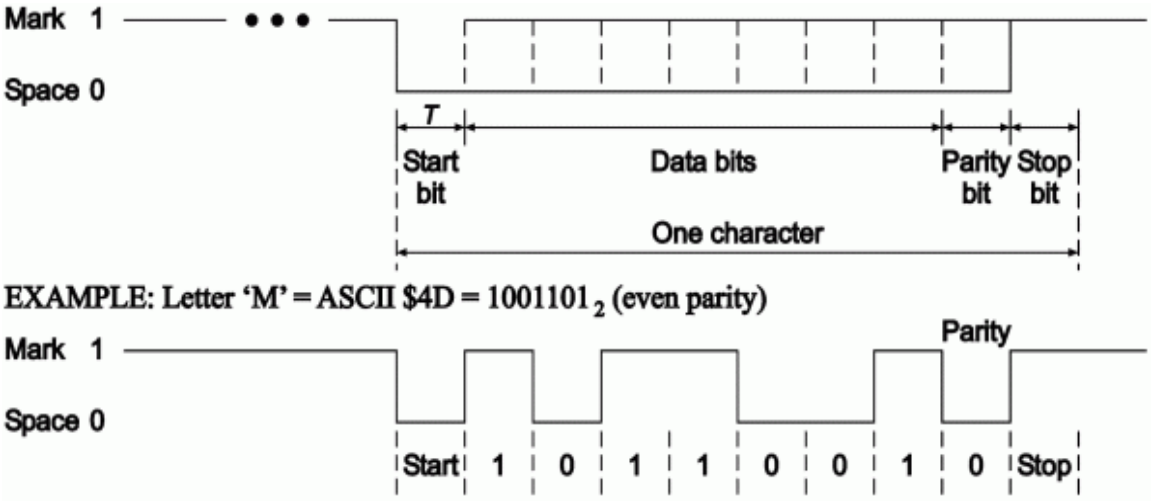


Figure 30

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

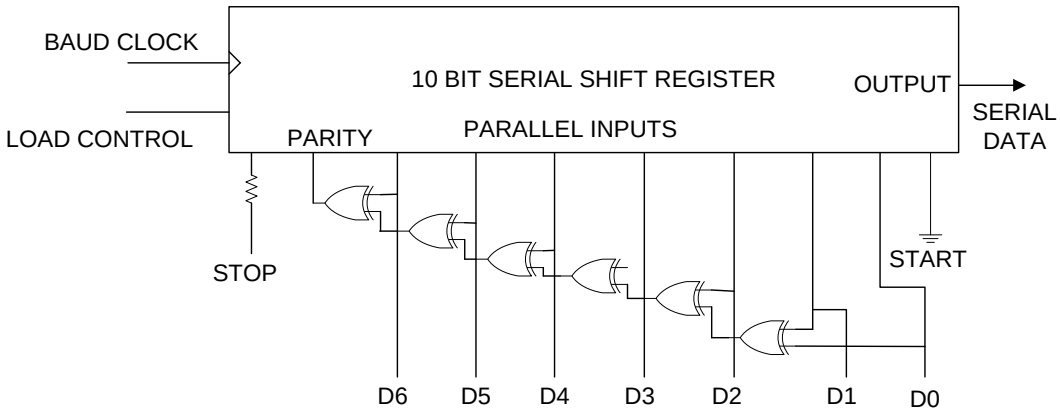


Figure 31

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

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).

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

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

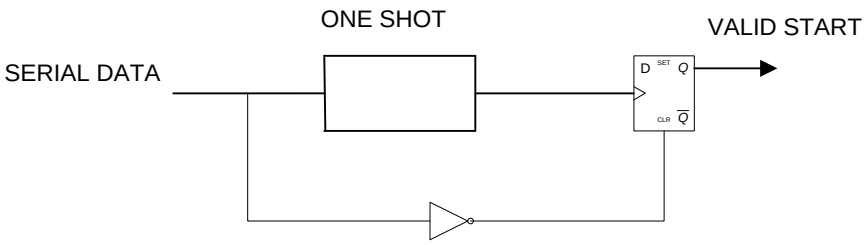


Figure 32

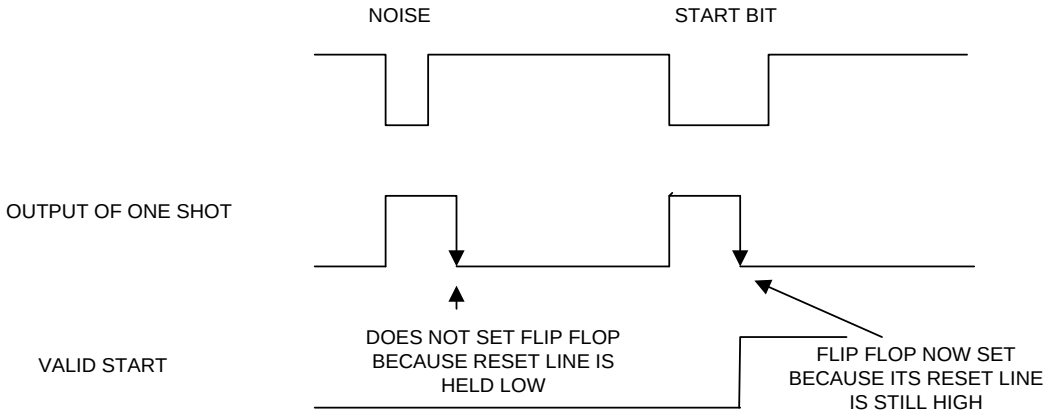


Figure 33

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.

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

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.

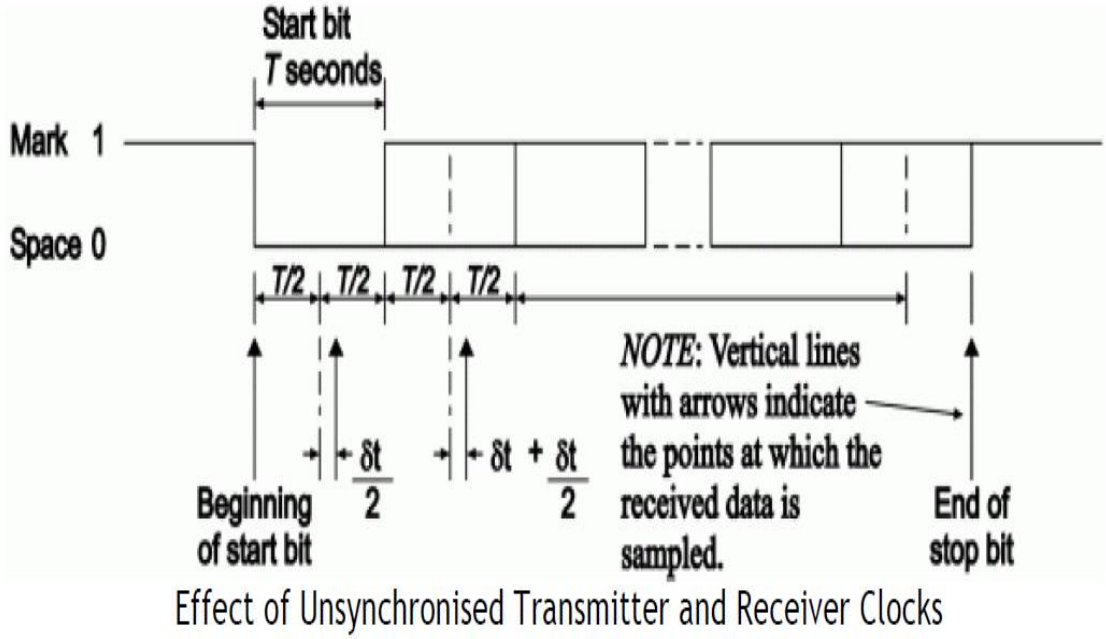


Figure 34

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

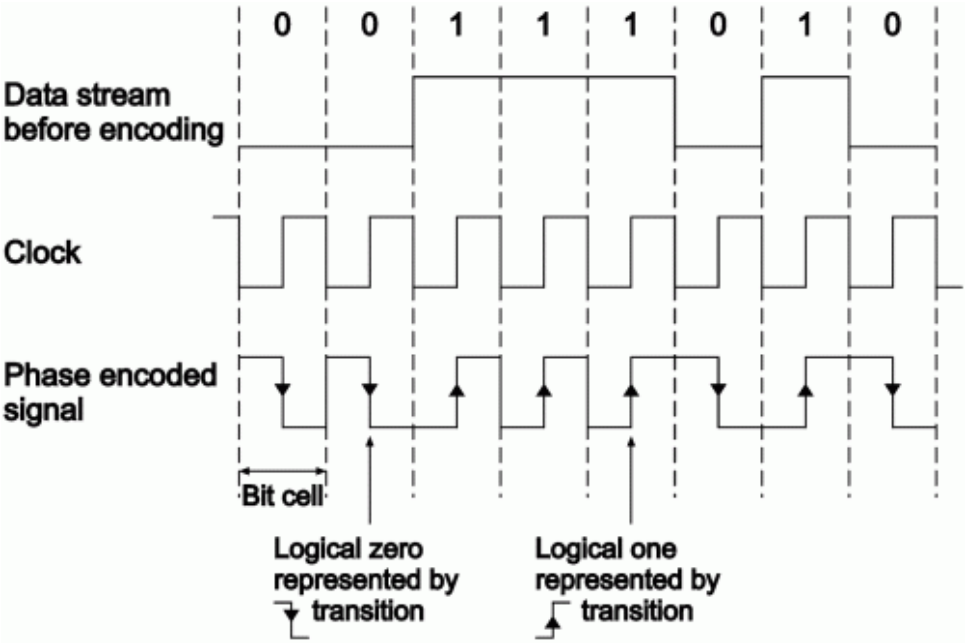


Figure 35

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

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

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

SECTION 10

MEMORY AND MEMORY TESTING

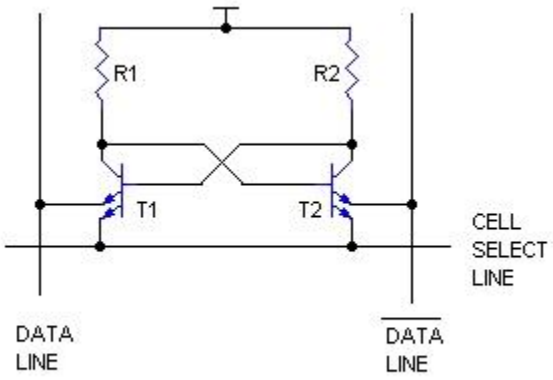
Random Access Memory (RAM)

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

Figure 36

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.

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

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. 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.

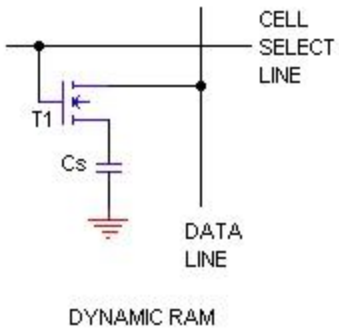


Figure 37

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 x Rin = 0.2 x 10^{-12} x 10^{10} x 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.

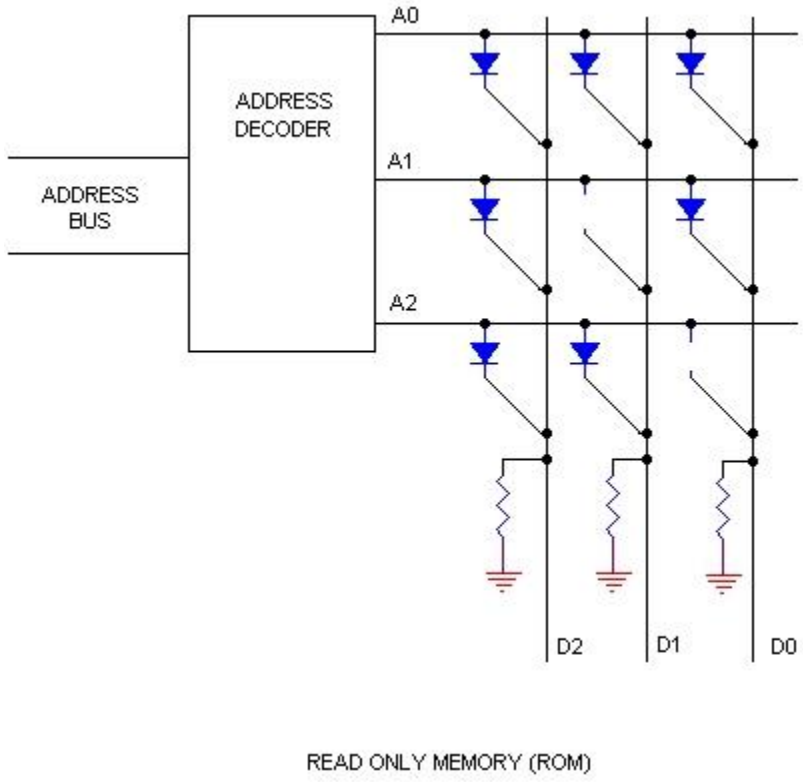


Figure 38

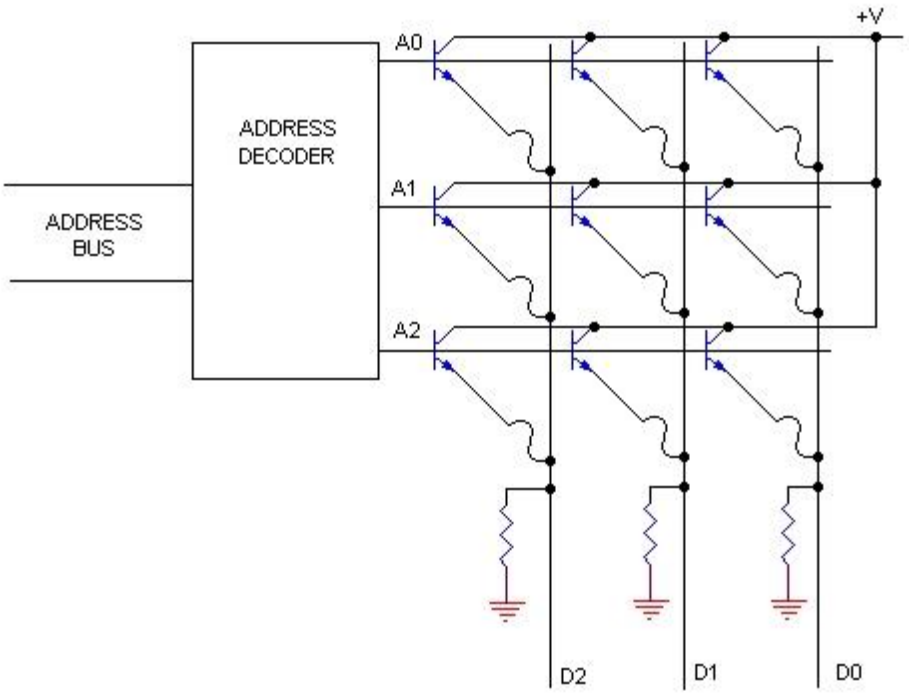
Lecturer: A Shanahan
Email: anthony.shanahan@dit.ie
Phone: 01 4024798

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)

Figure 39

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.

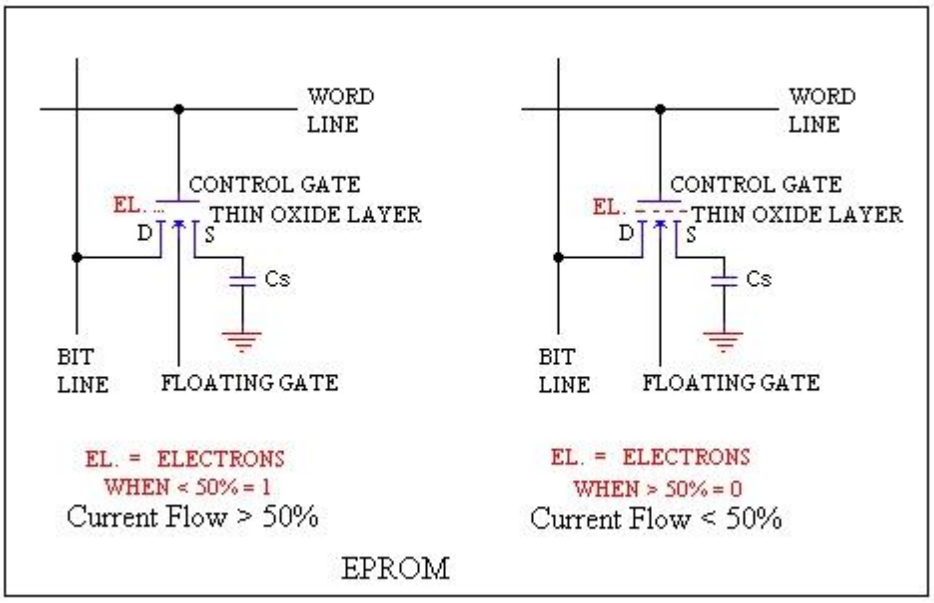


Figure 40

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.

SECTION 11

SAMPLE PAST EXAM QUESTIONS AND SOLUTIONS

NOTE: Some of these are from old exam papers and refer to the 6800/6809 microprocessors as used previously in the laboratory sessions. The parts that are not relevant are in italics.

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 a 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]

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.

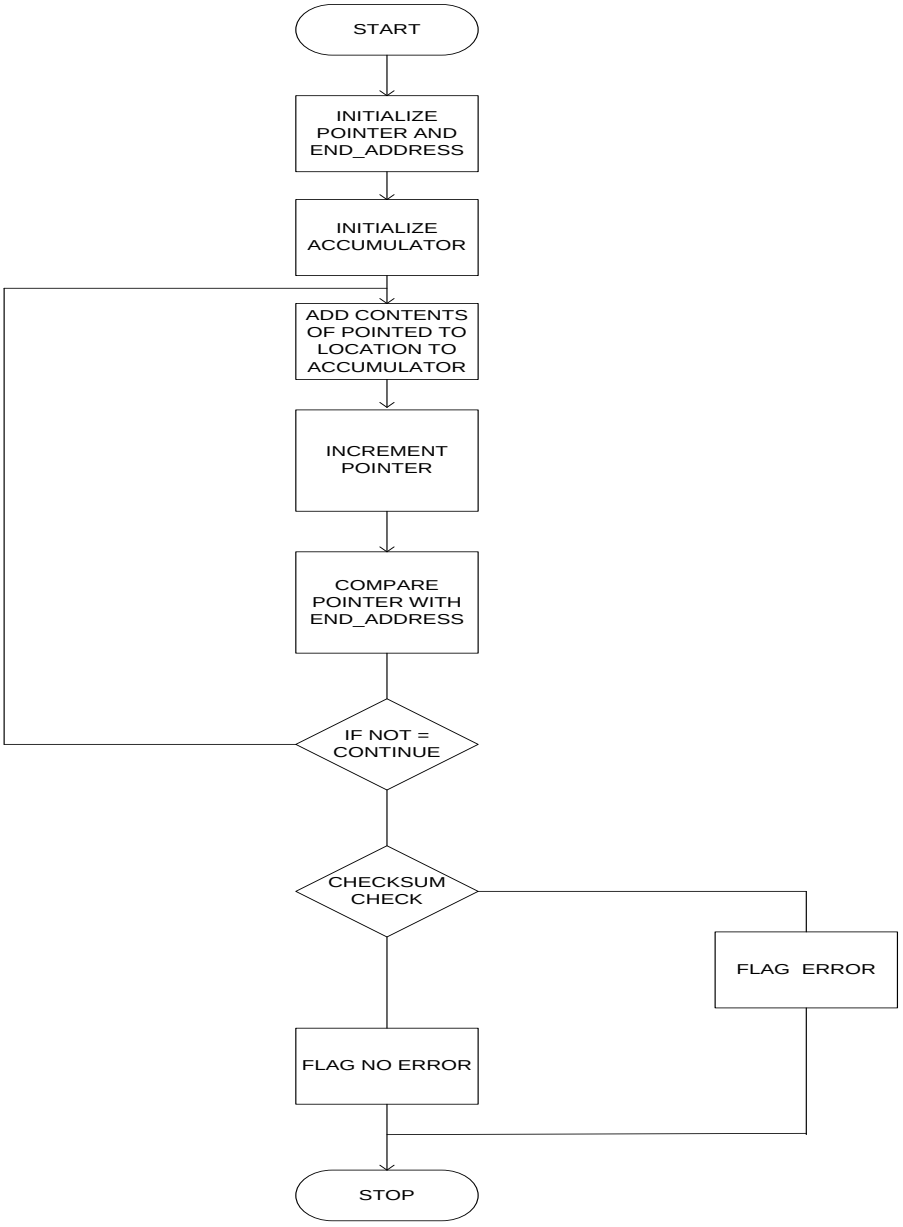
[12 marks]

SOLUTIONS

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

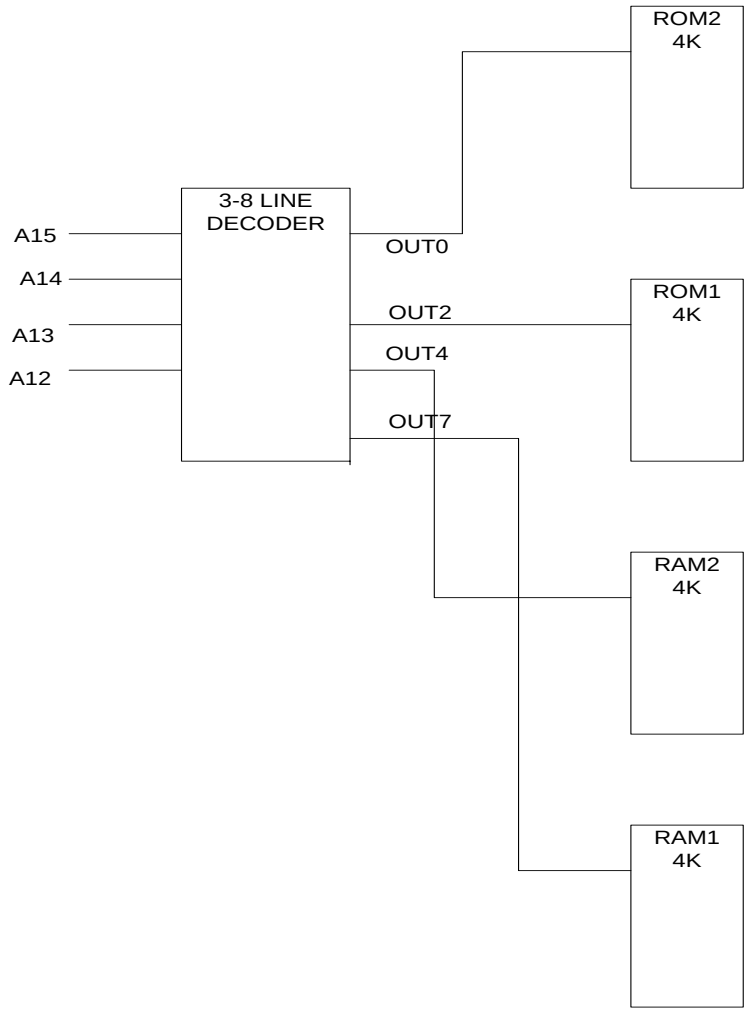
INPUTS			OUTPUTS							
S2	S1	S0	O7	O6	O5	O4	O3	O2	O1	O0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0

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 A15 can be used as the (active low) enable input, and A12, A13 and A14 can be used to select one of the four memory elements as shown in the block diagram below.



[16 marks]

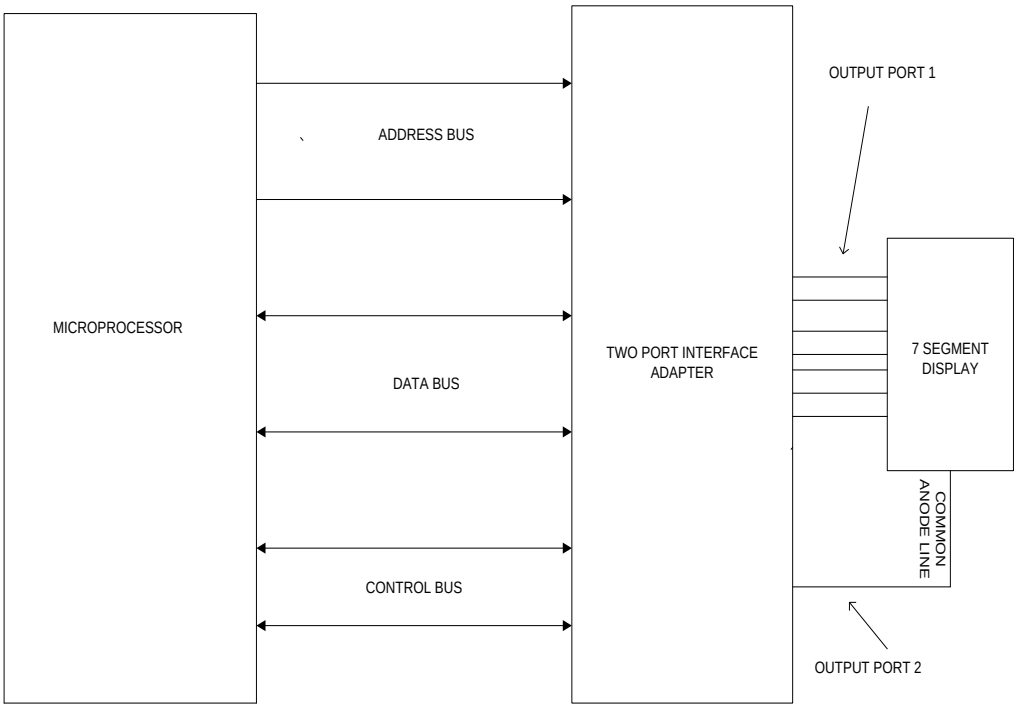
Q4 (a) In serial transmission data is sent one bit at a time. In order for the receiver to accept data synchronization between the receiver and transmitter is necessary. This is done with synch pulses in the case of synchronous transmission systems and with start and stop framing bits in the case of 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]

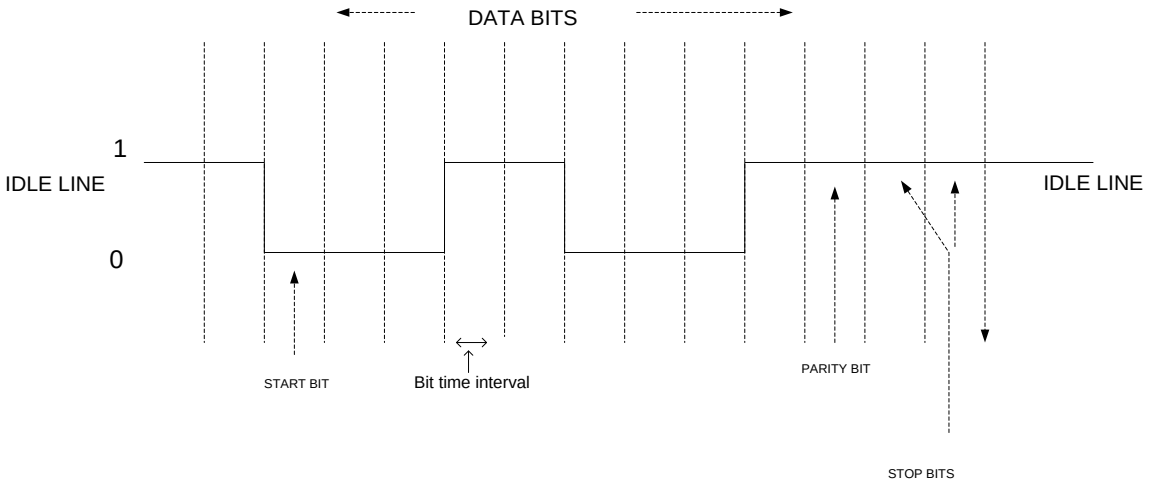
(b) Block diagram of display interfaced to a microprocessor:



[10 marks]

(b) 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]

SECTION 12

PIC 16F88 INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Instruction Word MSb			LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	lfff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	