Integrated Embedded Microprocessor Development System
Integrated Embedded Microprocessor Development System

Author

Mr. R. J. Spriggs

Course Title

Applied Computing and Electronics BEng(Hons)
Abstract.

The development of an Embedded Microprocessor Development System is discussed, drawing out the following aspects :-

- The design methodology and implementation used for the system and its subordinate products production.
- How the constraints of limited finance impinge on production of a cost effective and complete Development System Tool Set.

The novel aspects of this project include :-

- The concept of using Single Tool entities which can support a wide range of processors (Past, Present and Future) through self-contained dynamically linked libraries attached at run time.
- The use of Application Specific Languages for system customisation.
- Using Box Concept Design Method as the design tool.

The fully functioning parts of the system are :-

- The Integrated Development Environment for Software production.
- The General Purpose Cross Assembler.
- The General Purpose Macro Processor.
- The Limited Features General Purpose Object Code Linker.
- The General Purpose Processor Simulator.
- The General Purpose Device programmer operating in board level test mode.

The product set consists of both Software and Hardware designed entities of which many are capable of being used in a fully stand alone mode.

A Significant part of the Tool Set was used to develop the Device Programmer showing the viability of the product package.
Integrated Embedded Microprocessor Development System.

Table of Contents.

1.0 INTRODUCTION........................................................................................................... 6

2.0 SYSTEM DESIGN......................................................................................................... 8

2.1 Background.................................................................................................................. 8
  2.1.1 The Mini Computer Development ................................................................. 8
  2.1.2 The PC Based Development ........................................................................... 9
  2.1.3 The Dis-Assemblers ....................................................................................... 9
  2.1.4 The Cross Assembler ..................................................................................... 9
  2.1.5 The Library Manager ..................................................................................... 9

2.2 Recent Developments.................................................................................................. 10
  2.2.1 Limited Features Object Code Linker ........................................................... 10
  2.2.2 System Initialise and Integrated Development Environment ....................... 10
  2.2.3 General Purpose Simulator .......................................................................... 10
  2.2.4 General Purpose Macro Processor ............................................................... 11

2.3 Design Methodology................................................................................................... 12
  2.3.1 The Box Concept Design method. ................................................................. 12
  2.3.2 The Application Specific Language ............................................................... 12

2.4 Future Development.................................................................................................... 13
  2.4.1 Enhanced Linker Capability ......................................................................... 13
  2.4.2 Object Library Manager .............................................................................. 13
  2.4.3 General Purpose Dis-Assembler ................................................................... 13
  2.4.4 Common or Universal Assembler Code ....................................................... 13
  2.4.5 High Level Language Support ...................................................................... 13

3.0 SYSTEM SPECIFICATION. ......................................................................................... 14

3.1 User Requirements Specification. .............................................................................. 14
  3.1.1 User Assumptions. ......................................................................................... 14
  3.1.2 The Target Users and Requirements. ............................................................ 15
  3.1.3 Quality and Fitness of Purpose ..................................................................... 15

3.2 Software Development System Specification. ......................................................... 16
  3.2.1 The Cross Assembler Specification ................................................................ 16
  3.2.2 The Macro Processor Specification ............................................................... 17
  3.2.3 The General Purpose Linker Specification ................................................... 18
  3.2.3.1 The Limited features Linker .................................................................. 18
  3.2.3.2 The Full features Linker ....................................................................... 18
  3.2.4 The General Purpose Simulator Specification .............................................. 19
  3.2.5 The Integrated Development Environment Specification ......................... 20

3.3 Hardware Programmer Specification........................................................................... 21
  3.3.1 Introduction ................................................................................................... 21
  3.3.2 The Generic requirements. ......................................................................... 22
  3.3.3 The User Requirements. ............................................................................. 22
  3.3.4 User Interface .............................................................................................. 22
  3.3.5 Data Creation and conversion ..................................................................... 22
  3.3.6 Hardware Construction and maintenance .................................................... 23

3.4 The System requirements. ........................................................................................ 23
  3.4.1 The detailed device hardware interfacing ..................................................... 23
  3.4.2 The detailed processor hardware interfacing .............................................. 24
  3.4.3 The detailed software overview specification ............................................. 24

4.0 THE PROJECT PLAN................................................................................................... 25
Integrated Embedded Microprocessor Development System.

4.1 Project Activities.................................................................25
4.2 Programme Evaluation and Review Technique (PERT). ..................25
  4.21 Detailed Work Breakdown..................................................26
  4.22 Detailed Activity List Breakdown........................................26
  4.23 Estimation of work per activity.........................................26
  4.231 Identification of the basis of assumptions and estimations........27
  4.232 Resource Profile and impact upon estimated times................27
4.3 Risk Analysis...........................................................................28
  4.31 Risk categories and consequences......................................28
  4.32 Types of Risk.......................................................................29
  4.33 Causes of risk.................................................................29
  4.34 Constraints and their impact on risk....................................30
  4.35 Strategy for risk management...............................................31
4.4 Decision Analysis......................................................................31
  4.41 Improving decision making process....................................32
    4.411 Decision making and its impact on risk...........................32
  4.42 Decision tree .....................................................................33
    4.421 Forecasting ..................................................................33
5.0 HARDWARE DESIGN...............................................................34
  5.1 Design Plans..........................................................................34
  5.2 General Requirements........................................................34
  5.3 Initial Block Diagram..........................................................35
  5.4 Detail Block Diagram............................................................35
  5.5 Detailed Requirements and Implementation.................................36
    5.51 The ZIF Pin Interface.......................................................36
    5.52 The Voltage Generators....................................................39
    5.53 The Reference Voltage Generator......................................40
    5.54 The Voltage Monitors.......................................................40
    5.55 The Digital Interface.......................................................41
    5.56 The Process Control Interface.........................................42
6.0 SOFTWARE DESIGN..............................................................43
  6.1 Software for the Development System....................................43
    6.11 The General Purpose Cross Assembler.................................43
    6.12 The General Purpose Macro Processor................................47
    6.13 The General Purpose Linker..............................................49
    6.14 The General Purpose Cross Assembler.................................51
  6.2 Software for the Device Programmer.......................................54
7.0 USER INTERFACE.................................................................55
  7.1 The Software Interface........................................................55
    7.11 The Integrated Development Environment..........................55
    7.12 The Manual Method........................................................56
  7.2 The Hardware Interface.......................................................56
8.0 SYSTEM TESTING...............................................................57
  8.1 Hardware Testing Strategy....................................................57
    8.11 The Analogue Hardware..................................................57
    8.12 The Digital Hardware......................................................57
  8.2 Software Testing Strategy....................................................58
9.0 CONCLUSION............................................................................59
  9.1 The Successes.......................................................................59
Integrated Embedded Microprocessor Development System.

9.2 The Design Methods.................................................................59
9.3 The Hardware learning activities...........................................59
9.4 The Hardware Units..............................................................60
9.5 Future Development.............................................................60
9.6 Alternative Development Directions......................................61
  9.61 FPGA Processing...............................................................61
  9.62 Device Recognition............................................................61
  9.63 Device Emulation...............................................................61

LIST OF USEFUL REFERENCES.....................................................62

APPENDIX......................................................................................63
1.0 Introduction.

This dissertation will help to explain the methodology used for creating a fully Integrated Low Cost Embedded Microprocessor Development System. As a general rule, current electronics projects are most likely to involve a microprocessor and some discrete logic elements. The discrete logic elements may well be packaged into one or more programmed logic device/s. When a developer considers the implications of a new design, some of the major constraints are “the cost” and “the availability” of the development tools. In contrast, the advanced modern components remain relatively cheap when compared with the traditional approach components. The high cost of the tools and the experience required are major reasons why a developer will persist in using an older technology rather than innovate and use perhaps more appropriate and modern devices. An additional reason for remaining with the current development methods is that the developer is usually faced with a steep learning curve before he can progress with the project. Therefore, any system that can ease this learning curve and yet allow a rapid forward movement should be seen as an advantageous route to follow.

The Basic diagrammatic Summary of the solution view.
The integrated embedded microprocessor development system will address most of the concerns of developers embarking on using alternative products and currently consists of the following entities:

1. An Integrated Development Environment controlling the following items:
   - A Screen Text Editor.
   - A General purpose Text Macro Processor System.
   - A General purpose Object Code Linker System.
   - A General purpose Microprocessor Simulator System.


3. A General purpose machine that will be able to programme EPROM, PROM, embedded processors and associated logic devices such as Programmed Array Logic (PAL), Field Programmable Gate Arrays (FPGA) and Generic Array Logic (GAL). The primary hardware configuration is specifically focused on being able to programme just one of the Microchip PIC Family processor chips.


The Reverse Engineering Basic Solution Summary pictorial view.

4. Compile and Link

3. Modify the Source.

2. Disassemble the Code

Simulate the Code

5. Simulate the Code

6. Programme the device.

1. Read the device.
2.0 System Design.

The development of this System had a number of distinct focal points :-.

- The first part of the system development and production was driven by the immediate need to produce secondary products.
- The remaining development was to enable total product integration with a clear future progression path. A primary goal has always been to leave sufficient hooks in software modules and hardware development in order to allow, where possible, total flexibility of the product. This flexibility covers such areas as :-
  - Introduction of more Processors and Devices being added to the current product range.
  - Introduction of new or extended feature Directives.
  - Systems using a wide instruction code construct width.
  - Features such as Formal or Informal Macro processing.
  - The ability to customise the system with the “INIT.EXE” programme.
- All aspects of the System had to be highly Cost effective.

2.1 Background.

2.11 The Mini Computer Development System.

Some years ago, there was a requirement raised to develop a display system which incorporated a touch sensitive keyboard overlay. The system was to be based upon a Z80 processor. It was considered that it should be possible using the DEC PDP/11 MACRO assembly language to develop a series of macros that could mimic the Z80 instruction set. The Macros that were developed could then translate the Z80 mnemonics into a byte data stream.

The development of the Z80 based Macros generated the templates for a number of other assemblers such as 1802, 6502, 6800, 6809, 8049, 8080/5, Z8 to name but a few. The Macros operated quite effectively, however, they did run rather slowly. In the same period, a number of Dis-Assemblers were developed to match the Macro Cross-Assemblers, one to match each selected processor. It was the development of the Dis-Assemblers that indicated there was a significant amount of commonality between most processors. Hence it was only the unique processor code sections of the Dis-Assembler design that ever needed to be reworked.
2.12 The PC Based Development.
It was realised that any future product development would need to have an effective text editor. Rather than having to rely on editors produced and Licensed by third parties, a new multi feature editor would be created. Hence the first of the development tools was the Screen Editor (SED).
SED was targeted to have the following hybrid functionality :-
- The basic edit command sequences were based totally on one of the more popular PC text screen editors.
- Extensions were added to give almost all of the functionality that the DEC EDT editor supported.
The Screen Editor (SED) was the first of the Executable Shareware products.

2.13 The Dis-Assemblers.
The next phase was to translate the Dis-Assemblers into a language that could be run on a PC. As the original source code was BASIC-11, GWBASIC became a highly effective migration path with relatively minor amount of code change being needed. The modules were finally compiled using QBASIC for general distribution.

2.14 The Cross Assembler.
The next phase was Cross Assembler support. The original requirements analysis identified that the majority of functionality would be common to all processors. A number of specific entities were created, these being :-
- The Symbol Table Manager.
- An Expression Parser and Evaluator.
- An Object code Manager.
- A Source line Parameter Evaluator.
- A Instruction set Mnemonic Translator.

As the Instruction set Mnemonic translator would need to be unique for each processor, it was decided that this should be developed as an Interpreter controlled by some externally linked mechanism. To enable it to operate in the most efficient manner, the Interpreter was developed to trigger a nested Switch Case with pre-checked parameters targeted at pre-defined operation processes.

2.15 The Library Manager.
The externally linked mechanism is a fixed structure data file and was initially called a “Macro Library .MLB” because it was created from a series of pre-compiled macro type statements. It is now designated as an “Assembler Library .ALB” to avoid name conflicts with other libraries entities that are created by the (LMP) Library Manager Programme and other applications.
The Library Manager Programme reads the “Application Specific Language” code and generates a library file, or it can merge sets of pre-prepared libraries into a single entity. Hence the Interpreter in the Assembler became a variant macro processor. The system remained static with virtually no development for a significant period.
2.2 Current and Recent Developments.

2.21 Limited Features Object Code Linker.
As a result of once again looking at Embedded systems, the facilities of the Assembler suite have come sharply into focus. The Assembler produced a general hex binary format object code, however, this is unsuitable for downloading to such items as the Flight Boards. To overcome this problem, a Limited features Linker was produced. Its structure was based on the General Purpose Assembler using many of its common functions. The Linker supports the two most popular download formats, these being INTEL HEX and MOTOROLA S FORMAT plus extended features .HEX format.

2.22 System Initialise and Integrated Development Environment.
The **INIT.EXE** programme is common to all programmes used in the development suite. This programme is used to set up most of the control parameters and stores them in a file called ASS.INI in the working directory. This programme has had a significant number of recent updates or major reworks, some of these being:
- Introduction of a Multiple Configuration Registry.
- Enhanced to configure the Linker.
- Enhanced to configure the Macro Processor.
- Enhanced to configure the Simulator.
- Enhanced to be able to Control all the other Packages (IDE).

2.23 General Purpose Simulator.
This programme is a very recent addition to the overall system. The key requirements of this programme are to be able to:
- Simulate a range of processors (Past, Present and Future).
- Simulate devices attached to the processor.
- Be configured with different hardware options.
- Record or log simulation progress.
- Stimulate the simulator with a control script.
- Monitor / Time or Interrupt progress as the user requires.
- Operate on a minimal Hardware configuration.

Again due to the nature of the requirements, this programme has many features in common with the General Purpose Assembler. To cope with the requirement of extendible processors or devices, a similar methodology has been used in the form of an interpreted “Application Specific Language”. The Library Manager Programme (LMP) compiles the code into a dynamic Simulator Linking Library (a .SLB file).

Due to a wide possibility of requirements which may be needed to simulate different processors, the simulator had to be made as flexible as possible. However, some restrictions have had to be placed on Simulator and these are:
- Any unique address space is limited to a 32 bit address.
- The maximum width of a single instruction is limited to 128 bits.
- A single processor and its devices are limited to 16 unique address spaces.
- The maximum system map space is $2^{31}$ locations.
2.24 General Purpose Macro Processor.
This programme is a very latest addition to the overall system. The basic Assembler is very successful at processing single line statements and providing the number of bytes created by a statement does not exceed sixty four, there is fundamentally no problem. However, there is a number of situations where data or programme structures logically span across a number of lines. A second scenario is where calculations need to be performed upon and within sequences of data. The final possibility is a structure development where both previous requirements occur.
One solution to the requirement is a source code embedded pre-process editor. However, for total flexibility, the ideal solution is a full features Macro Processor.

So what is a Macro Processor and what does it do?

At the simplest level, a Macro Processor behaves like a simple text editor substituting one series of character for another sequence. However, a Macro Processor should contain sufficient control structures to allow a significant level of intelligence to be applied to the dynamic editing process.

The intelligence is usually implemented by using structures like :-

- {IF <Condition (True/False)> [Create Block of Text]}
- {REPEAT <Number of times> [Create Block of Text]}

The next important feature that gives the intelligence to the process is the Block of Text that accepts parameters which can be substituted into itself (i.e. A Macro). A block of text is defined with embedded markers showing where the parameter substitutions should occur. Each time the Macro is exercised, the new parameters are substituted creating text block conforming to a template as defined by the Macro.

It is evident that the overall Macro expansion process can be quite complex and convoluted, especially when we consider that Macros can also call other Macros recursively. They can also contain at all levels both Conditional and Repeat text blocks. It should not be forgotten that Macros usually have the ability to develop dynamic parameters, further adding to the expansion complexity.

So why do we want Macros?

Quite clearly from the previous description, the creation of Macros can be very complex and debugging them could be a problem. However, in general, there is a number of reasons for using Macros :-

1. Macros can simplify and significantly speed up programme development.
2. Macros can be reusable.
3. Macros can make code Platform Transportable.
4. Macros can make code much easier to read.
   e.g. PRINT “Have A Nice Day”
   It will not take much effort to guess what the above does.
2.3 Design Methodology.

There have been a significant number of Design Methods proposed for Software and Hardware design over the last few decades. UML, YORDON, SSADM, HIPO and Flowcharts to name just a few. However, many of these systems are simply variants on a theme with perhaps the major difference being the diagram symbols. For the purpose of this document, a further two variants are to be added to the list and have been used to develop the majority of this “Development System”. These are:

- The Box Concept Design method.
- The Application Specific Language.

2.31 The Box Concept Design method.

This Design method is a variant of the Block diagram. The System, Sub Systems or lower level entities are defined as a Box or series of boxes. Each Box or Block is then examined in more detail and is either:

- Further decomposed in a set of lesser Boxes.
- Has a description attached to the block describing its function.
- Has a description attached to the block describing its contents.
- Has a description attached to the block describing its relationship to other entities.

The primary advantages of this system are that:

- It is very easy to implement.
- The diagrams are fairly intuitive to read or describe.
- Arrows can be added to identify flows or relationships.
- No specialist diagram drawing tools are needed.
- The level of decent detail can easily be adjusted to the presentation or design requirements to give a clear focus point.

2.32 The Application Specific Language.

For some years, there has been a trend to develop computer languages that try to be universal (one language for all applications). Typically “C”, “C++”, “C#” and “JAVA” fall into this category. They have many programme constructs and data types with the opportunity to develop user defined structures. An alternative approach is the “Application Specific Language” which is directly focused at producing one type solution for a limited range of problems. The “Application Specific Language” will tend to have the following features:

- The Language writing style is likely to be unique.
- Have few or NO definable Data types.
- The programme code can be directly generated from the problem domain.
- The programme code is also the problem solution documentation.
2.4 Future Development

2.41 Enhanced Linker Capability.
The Linker currently only supports absolute binary object images. The next major development phase will include: -
- Re-locatable binary object file support.
- Object Library support.
- Multi Region Image building.
- Checking of duplicate location usage.
- Additional Binary Image Formats.
- Enhanced more meaningful Map analysis.

2.42 Object Library Manager (LIB).
Even from the early days of computing, a need has been seen for the ability to maintain pre-compiled object modules. The Assembler already supports hooks for creating relocatable object modules. The Linker has a menu to support the inclusion of External Libraries into the current build. This will be the next development product.

2.43 General Purpose Dis-Assembler.
This programme will replace all the single processor Dis-Assemblers and the common programme DISEDIT, however, it will give an identical output to the programmes it replaces. It will have a similar front end to the General Purpose Assembler and will select a .DLB Dis-Assembler library to perform the Dis-Assembly process. The .DLB file will be created by (LMP) the Library Manager Programme. The development of this product will basically consist of a collation exercise of pre-existing items and a small amount of integration code.

2.44 Common or Universal Assembler Code.
It has been considered that as many processors have a similar capability instruction set, it may be possible to develop a Common Assembler Code that can be translated or converted to run any processor. This would then only require a few new libraries to be created to give the extra functionality.

2.45 High Level Language Support.
A further level of functionality that could be offered to the suite would be the ability to compile sources in say “C”, “PASCAL” or “BASIC”. The Compilers would then translate the original source codes into an intermediate format which could be converted into an appropriate object code by the Assembler and the selected processor library. The final stage would be to Link the Objects to give the runable binary image.
3.0 System Specification.

The System Specification is separated into the following areas :-
- User Requirements Specification.
- Hardware Programmer Specification.

The main reason for part of the separation is that a fully functional and operational Software Development System was a prerequisite for the development of the Hardware Programmer. The Software Development System is required for producing the Drivers and Application Programme for the Hardware Programmer. It is also needed and used for all of the Test and Commissioning software.

3.1 User Requirements Specification.

This section will cover the following topics :-
- User Assumptions.
- Target Users and Requirements.
- Product and Quality.

3.11 User Assumptions.

The system is targeted at users who :-
- Are working to a limited budget but still require product flexibility.
- Have a working knowledge of the device/s they intend to use.
- Are likely to require to work on more than one processor family.
- Are willing to read the user manual rather than expect the system to run in automatic user teaching mode.

The system has a number of user interface modes such as :-
- The Integrated Development Environment (IDE). This mode is targeted for the new or inexperienced user who may be unfamiliar with driving a system from a command line prompt.
- Command line Mode which is for the experienced user.
- Stand Alone mode for applications that only use part of the system suite.
3.12 The Target Users and Requirements.

The system will have an appropriate interface to suit the ability of most users and is particularly focused towards those who are likely to have limited resources. However, there are users whose requirements are so special that a bespoke solution would normally only be available to organisations with significant funding.

The typical target user groups are :-
- The Hobbyist.
- Education Establishments.
- Organisations with limited resources.
- Private System or Specialist Developers.

The user that requires the bespoke solutions would require two additional programmes currently NOT part of the shareware package, these being :-
- The Library Manager Programme (LMP). This programme allows the user to develop his own specific application solutions to his own requirements. This programme also allows pre-compiled libraries to be linked into a single entity and supports a Library Language Word usage Analyser.
- The Library Scan Programme (LSP). This programme allow patches and minor modifications to be made to libraries. It is also used as a Library debugging tool, and Library Dis-Assembler.

3.13 Quality and Fitness of Purpose.

The system quality checking and fitness of purpose is implemented in the following manner.

The system will support :-
- Test suites issued with the system so that the user can verify that its functionality conforms to specified requirements. The test suites are used for quality checking.
- Default typical system configurations that can be altered to the User specification.
- Options that will allow the user to be able to build the hardware with the specific functionality he requires.
- A System User interface style common across all packages.
3.2 Software Development System Specification

This section will cover the following entities:
- The General Purpose Cross Assembler.
- The Macro Processor.
- The General Purpose Linker.
- The General Purpose Simulator.
- The Integrated Development Environment.

3.21 The Cross Assembler Specification

The Cross Assembler will have the following features:
- The ability to be controlled by:
  - Command line message.
  - Menu driven interface.
- The ability to process a “Main Source” that contains “Include Files” directives.
- It can directly activate the Macro Processor with a pre loaded source image.
- It can be configured to support a range of Processor mnemonics.
- It can be customised for:
  - Optimum Workspace usage.
  - Pre-selected Default Settings.
- It can support:
  - Relocatable binary images.
  - Both Numeric and String Symbols.
  - 16 or 32 bit calculation mode.
  - 16 bit High/Low byte reversal display option.
  - Defined symbol name widths of 6 to 24 characters.
  - Positive fraction constants.
  - ASCII character value conversion constants.
  - 8,16 and 32 bit data storage directives.
  - Radix base 2,8,10 and 16 defined values.
  - Local and Global Symbols constructs.
3.22 The Macro Processor Specification.

The Macro Processor will have the following features :-

- The ability to be controlled by :-
  - Command line message.
  - Menu driven interface.
- The ability to Process a “Main Source” that contains “Include Files” directives.
- It can be customised for :-
  - Optimum Workspace usage.
  - Pre-selected Default Settings.
- It can be operated
  - In Stand Alone mode.
  - As a slave process of the Cross Assembler.
- It can transfer Symbol definitions to the Cross Assembler
- It can support :-
  - Macros with or without Parameters.
  - Macros with (dynamic and default parameter) substitution.
  - Conditional assembly “IF” and nested “IF” blocks.
  - REPEAT and Nested REPEAT Code blocks.
  - Global (dynamic and default parameter) substitution.
  - Formal and Informal Macro Expansion mode.
3.23 The General Purpose Linker Specification.

This product will be developed in a number of phases:
- Phase One The Limited features Linker.
- Phase Two The Full features Linker.

3.231 The Limited features Linker.

This entity has very limited processing capability. It will only process Binary Object files that have all its symbol references resolved.

This Linker will have the following features:
- The ability to be controlled by:
  - Command line message.
  - Menu driven interface.
- The ability to process a:
  - Single Programme Object Source.
  - Definitions file containing a list Programme Object Sources.
- It can be customised for:
  - Optimum Workspace usage.
  - Pre-selected Default Settings.
- It can produce as an Output file:
  - INTEL HEX Binary Images.
  - MOTOROLA S Binary Images.
  - Simulator Extended HEX Binary Images.

3.232 The Full features Linker.

This Linker will have the following features:
- All the features of the Limited variant specification.
- The ability to:
  - Process relocatable objects files.
  - Support Link Library files.
  - Support multi region images.
- It will identify and error report binary image data clash conflicts.
- It will produce a detailed Linker analysis map output.

The General Purpose Simulator will have the following features :-
- It will be controlled by Menu Driven Interface.
- It will be able to be customised for :-
  - Optimum Workspace usage.
  - Pre-selected Default Settings.
- It will be able to be configured :-
  - To support a range of Processors and Devices.
  - With Memory Regions set to user requirements.
    - Memory will be able to be :-
      - Mirrored.
      - Mapped to Alternative Regions.
      - Have READ Only access.
      - Have WRITE Only access.
      - Have READ and WRITE access.
      - Have Independent READ and WRITE access at a common data address.
- It will support loading of one or more Simulator Extended HEX Binary images.
- It will have Control Mode support for :-
  - Single Processor Simulation operation mode.
  - Single Processor and Device Simulation interaction environment operation.
  - Master Processor and Slave Processor Simulation interaction operation.
- The Operation Command modes are :-
  - Instruction Single Step
  - Continuous Run
  - Run for a defined number of Instructions
  - Run ignoring any Simulator System Data Access Errors
- It will be able to generate a run time report log to:-
  - A Log Display screen.
  - A Data file.
- It will be able to :-
  - Run timed command script files.
  - Produce a run time code Dis-Assembly.
  - Modify and examine data images under user control.
  - Display a Pseudo Device Animation Screen.
- It will be able to simulate :-
  - DMA transfers.
  - Interrupt operations.
  - I/O interface device interaction.
  - Instruction sets widths up to 128 bits.
  - A maximum of 16 memory space regions.
- All Simulator Regions will have a 32 bit address space limit.

The Integrated Development Environment (IDE) is a sub entity of the system initialise or configuration programme (INIT.EXE). The INIT.EXE programme can be controlled either by command line prompts or a menu driven interface. The method of operation is purely a matter of choice. see file USER.MAN for details of the various implementation activation options.

The Integrated Development Environment will have the following features :-

- It will be controlled by a Menu Driven Interface.
- It will be able to initiate :-
  - The workspace usage of other programmes in the suite.
  - The customised Default Settings for other programmes in the suite.
- It will be able to :-
  - Verify that processing programmes are available for use and error report if the programmes are unavailable.
  - Select a default preference Editor programme.
  - Display contents of User manual file.
  - Create a basic Assembly Programme Source Template.
  - Activate or Start the Assembler and Linker process.
  - Display the Current Programme Listing output.
  - Support Multi Project working environment by :-
    - Changing the State of the Current Environment.
    - Saving then Current Environment in a named Registry area.
    - Restoring an Environment from a named Registry area.
  - Enable the Simulator Configuration file to be edited.
  - Display the Current Simulation Library documentation file.
  - Activate or Start the Simulator.
3.3 Hardware Programmer Specification.

3.31 Introduction.

There are hundreds of different field programmable devices types. The specialist equipment needed to support these devices can be very expensive. This is in part primarily due to the fact that many technologies are used to develop the devices and hence the potential for each device gives a limited user base. There is little in the way of a universal hardware programming standard. This is mainly due to the unique nature and construction technologies of the devices. The number of devices available in the market place which can be currently programmed is in the order of hundreds of thousands with more new devices becoming available every day.

The General Purpose Device Programmer was conceived to enable those on a limited budget to enter the programmable device market. These new users will now be able to take advantage of these devices. Furthermore, it will give these innovators the same advantages of electronic circuit designs and products as those who are working in larger and better funded organisations.

The remainder of this section will be subdivided into the following subsections :-

1. The Generic requirements.

2. The User requirements

3. The System requirements.

   a) The detailed device hardware interfacing.

   b) The detailed processor hardware interfacing.

   c) The detailed software overview specification.
3.32 The Generic requirements.

Typical requirements of a General Purpose Programmer Device are :-
1. It needs to have a fast, easy and effective method to “connect to” and “remove from” the Device and the Programmer.
2. It may need to be able to supply power to the device that is to be programmed.
3. It may need to be able to supply logic levels to device pins.
4. It may need to be able to supply programming voltages to one or more specific device pin/s. (The programming voltage in this context means a voltage which is different from the devices normal operating range).
5. It may need to be able to read status and verification information from the device being programmed.
6. It may need to supply phased clock signal to specific device pin/s.
7. It will need to ensure that the voltages it supplies to a device remain within the specification of that device being processed in order to maintain quality production.

As there is no particular standard pin layout, all the desired features for programming a device will need to be available on every device pin. However, if a user wishes to limit himself to a particular range of devices, this may allow the overall system functionality to be downgraded to a more appropriate level.

3.33 The User Requirements.

3.331 User Interface.

The GPDP operations will be controlled either via a menu driven interface to a host machine or in a pure “turn key” mode for production work.

The menu driven interface will have two distinct user operating modes :-.
- New user Mode will lead the user step by step through all the appropriate operation stages to get a successful outcome.
- Advanced user Mode will allow appropriate shortcuts to give the desired outcome.

The “turn key” operation mode would typically be :-
1. The System will indicate it is ready for “Device load” or ”Programming complete”.
2. The user inserts a device in programming socket and then presses programme start button.
3. The System indicates that the Programmer is “busy”.
4. On completion, the programme indicates the operation was either “Success” or Failure”.
5. The user removes the device and records programming status. (Back to Start)
3.332 Data Creation and conversion.

The Integrated Development Environment (IDE) will initially only be suitable for developing binary images for microprocessors. The IDE is already capable of generating binary images suitable for loading into other systems. It is not unreasonable to expect that the user may wish to continue using tools he is already familiar with which generate binary images. If the user has already been supplied with binary images developed by other tools, he may now need to programme chips from the supplied images. Therefore, the GPDP will not only accept binary formats from the Integrated Development Environment (IDE) but will also accept most of the industry standards i.e. JEDEC etc. However, if too many formats are presented, then a format conversion module may need to be embedded in the IDE to resolve any overload conflicts.

3.333 Hardware Construction and maintenance.

The General Purpose Device Programmer (GPDP) will be designed using a fully modular approach. This will enable :-
1. The GPDP user to start with a limited features machine and expand it as their budget permits or programming needs and requirements change.
2. Low cost specialist device GPDP could be created for the production environment.
3. Sub modules can be built, tested independently or even sold as a kit of parts.
4. In the event of a sub module failure, maintenance or replacement will be an easy process.
5. Franchise, production, maintenance or development could be offered to appropriate organisations.

3.34 The System requirements.

3.341 The detailed device hardware interfacing.

The design of the General Purpose Device Programmer programming module will :-
1. be able to be controlled from a series of parallel TTL Level Interface signals.
2. be expandable in steps of 8 Pins to a maximum of 64 Pins.
3. attach all device pins via a ZIF (Zero Insertion Force) Socket (maximum size 64 Pins).
4. support a fixed “default +5Volts source” to any pin.
5. support changing the “default +5Volt source” with a supply from a digitally controlled analogue voltage generator with a range of 0 to +25Volt in 64 or more incremental voltage steps.
6. support device pins being used in a high speed clock mode.
3.342 **The detailed processor hardware interfacing.**

The Hardware design requirements of the General Purpose Device Programmer Processor will include :-
1. A common interface interconnection to the General Purpose Device Programmer module.
3. An optional USB master control interface to be able to replace the RS232 serial interface.
4. An optional Flash Memory Disk Interface.
5. A battery backup real time clock to enable automatic date and time stamping.
6. An onboard configuration memory store.
7. A optional direct connection programme configuration and data entry interface for stand alone operations.

3.343 **The detailed software overview specification.**

The Software design of the Overall General Purpose Device Programmer will eventually incorporate such features as the ability to :-
1. automatically detect its own hardware configuration.
2. regularly check its own basic calibration with a log of its status.
3. keep an upload quality assurance log of its programming activities with the outcomes.
4. accept downloads of programming methods.
5. hold a range of direct select preferred programming methods.
6. hold a range of direct select preferred programme images.
7. verify that its own hardware configuration is compatible with the device being requested to be programmed.
8. Verify that data and protocol downloads have not been corrupted.
9. Verify that downloaded protocols have not date expired. (Author comment: “It has been noticed that as manufacturers develop or enhance their products, the recommended algorithms necessary to programme their devices may alter. This is a method that could catch obsolete algorithms. However, support of past algorithms may be necessary if batches of the older device still exist in the market place”.)
4.0 The Project Plan.

4.1 Project Activities.

The initial development stages viewed the whole project and decomposed it into a number of main identifiable activities. These were :-

- Prepare the Development System.
  - This would consist of the Assembler, Simulator and other Programmes.
- Develop System Hardware.
  - This is specifically the Programmer developed in a modular manner.
  - It was conceived as consisting of :-
    - A Processor Board/Section.
    - A Main Interface Board/Section.
    - A number of sub function control Board/Sections.
- Develop Main Processor.
  - This processor would perform the following activities
    - Act as interface between the programmer and the Host System.
    - Control the functionality of the programmer.
    - Would perform application specific activities.
- Develop System Software.
  - This is the Software of Main Processor.
- Develop Mother Board.
  - This would be the Main Programmer Control and Interface Board.
- Develop Daughter Boards.
  - These would add the overall functionality to the Mother Board.
- Feasibility Study.

4.2 Programme Evaluation and Review Technique (PERT).

This section will cover :-

- Detailed Work Breakdown.
- Detailed Activity List Breakdown.
- Estimation of work per activity.
- Identification of the basis of assumptions and estimations.
- Resource Profile and impact upon estimated times.

The following items are covered in Appendix PERTCALC.doc

- Layered PERT Diagram.
- Calculations of estimated times and variances.
- Estimated time for completion.
- Calculations of probabilities of project overrun.
4.21 Detailed Work Breakdown.

The production of a detailed work-break-down chart was not possible as this feature has been removed from the processing package. However, the detailed PERT diagram gives the same information and has been structured to show breakdown and linkages of the following top level development work package tasks:-

- Prepare the Development System.
- Develop System Hardware.
- Develop Main Processor.
- Develop System Software.
- Develop Mother Board.
- Develop Daughter Boards.
- Feasibility Study.

4.22 Detailed Activity List Breakdown.

The detailed activity lists are shown in :-

- Appendix Filename PERT_EIE.XLS.
- Appendix GANTT Chart.
- Appendix PERT Diagram.

and is segmented as follows :-

- Prepare the Development System.
  - Entries 2 to 9.
  - Entries 11 to 18 resulted from need to rebuild a previously designed processor monitor programme.
- Develop System Hardware.
  - Entries 21 to 22.
- Develop Main Processor.
  - Entries 23 to 27.
- Develop System Software.
  - Entries 29 to 34.
- Develop Mother Board.
  - Entries 36 to 38.
- Develop Daughter Boards.
  - Entries 40 to 49.
- Feasibility Study.
  - Entries 51 to 52.

4.23 Estimation of work per activity.

The document Appendix Filename PERT_EIE.XLS shows the estimates in man-days for the most likely, optimistic and pessimistic times for each activity.
4.231 Identification of the basis of assumptions and estimations.

Many of the assumptions that have been made are based on past experience of similar work, however, there is always an element of educated guess. This becomes a particular problem when working in unknown areas. In these cases, there is a tendency to err on the side of caution especially when dealing with entities that are known to be potentially complex. It is the complex sub sections’ duration which is the hardest to predict due to the potential internal and external interactions. This is particularly so with the layout of printed circuit boards. When a small number of components is required, this presents only a minimal problem i.e. a few hours of layout at most. However, once the number of through board holes approaches 200, significant difficulties are usually experienced with the layout. The auto routing and auto placement options may help, however, they usually route the power lines first, resulting in an excess of vias between layers of the signal lines. This creates its own unique problems when it comes to board production i.e. the accuracy required to place the vias and the time involved in soldering them in place. When laying out the PCB, the manual approach would be to keep as many tracks as possible on the copper side of the board and to keep the via count to a minimum. These techniques do require a certain level of intuition as ideal component placement is paramount. If these constraints are not enough, the designer will also need to take into account component direction placement for ease of construction and track widths for power distribution.

4.232 Resource Profile and impact upon estimated times.

The decomposition of the project has identified that there is a number of parallel work paths and normally, different sections of the project would be allocated to different individuals or groups of individuals (Human Resources). The resources could consist of both internals staff and persons contracted specifically for certain activities. A second group of resources that needs to be considered is equipment, workspace and services that will need to be available for completion of the task. The non human resources may also raise allocation and requirement conflicts. The staff using these shared resources therefore need to compromise with other staff when making use of limited facilities. It may be necessary to arrange specific scheduling of some resources to overcome the allocation overloads. The result of the rescheduled work is usually to extend the estimated time of the planned requirement for a specific task. This becomes all the more important if these specific tasks are on the critical path. The resource profile for this specific project causes one of the major problems. There is only one resource to complete all activities and hence by definition all activities must be on the critical path no matter which parallel route is being examined. Normally, a project would have more than one resource and critical path becomes more meaningful. The critical path is therefore by definition in this specific case the sum of all the working time for all projects.
The resources available to the project are rather limited and hence effective use of available resources is paramount. For the majority of time, there is only one individual to complete all the activities. However, there will be a significant number of activities that could sensibly run in parallel. There will be a few activities that will require a small amount of third party assistance or safety monitoring. These activities specifically are:

- Printing of photo masks for PCB manufacture.
- Supply and selection of PCB board material.
- Arranging appropriate access times for the PCB manufacture.

4.3 Risk Analysis

This section will cover:

- Risk categories and consequences.
- Type of Risk.
- Causes of risk.
- Constraints and their impact on risk.
- Strategy for risk management.

4.31 Risk categories and consequences.

Risks basically fall into three categories:

Type 1 Normal or typical day to day problems.
These are the common occurrences and will be typically covered in the allocation of the pessimistic prediction of task duration.

Type 2 Rare events.
These are derived from investigating assumptions made about the project. They may need some level of contingency planning and perhaps insurance to hedge against potential loss or incurred penalty expenses.

Type 3 Project fails, Never finished or Rejected by sponsor (Act of God).
This is the case when the project delivers nothing so all investment is lost. The level of failure may also incur penalties due to lack of delivery. So not only is the investment lost but the penalties may force the provider into receivership.

This project will only consider the implications of Type 1 risks as Type 2 & 3 are more appropriate to a commercial organisation. In the event of a Type 2 or 3 occurrence, then it is back to square one. Start again with a new project or salvage redefined elements of current project.
4.32 Types of Risk.

The types of risks for this project can be subdivided into distinct areas and will require different strategies to achieve a successful outcome specifically:

- The risks associated with producing a final output from the project.
- The risks associated with the product being fit for purpose.

The Risk management strategies used to produce a final output from the project will be resolved by:

- Making effective use of all available resources.
- Have an alternative approach option for all high risk activities.
- Keeping Backups archives of documentation and planning activities.
- Monitoring Progress, Finance, Resource allocation and ensuring all Milestones achieved on Time and within the Budget allowed.

The Risk management strategies used to ensure the product is fit for purpose will be resolved by:

- By using incremental development and ensuring that all prototype interfaces meet the Customer requirements.
- The System interfaces are easy to use and are supported by clearly defined procedures and User documentation.
- The System is incrementally validated against the specification ensuring that each subordinate entity is fully functional before submission to full system integration.
- The system will be developed ensuring that all appropriate current Safety features and Radiation shielding requirements are included within the product.

4.33 Causes of risk.

The most significant risk and problem areas of this project will be:

1. Getting the programmer to be recognised by chip manufacturers.
2. Finding a highly cost effective plug and socket system for sub module docking.
3. Safety issues of using a programmer without a case. The board is a low voltage device, however, desk contamination could damage the programmer rather than being a particular threat to the user.
4. System reliability due to high component count.
5. On the development front the main risk areas will be:
   a) Large PCB Layouts (Processor Board and Mother Board).
   b) Complex Application Software programme implementation.
   c) Development of an Integrated Macro Processor programme.
   d) Component sourcing as many standard items are NOW only available in surface mount format.
4.34 Constraints and their impact on risk.

The major constraining factor of this project is COST and has the most significant effect of possible development routes. Due to lack of funding, the surface mount production option is effectively shut off (cost of purchase of equipment). Long term, this will have a significant effect mainly due to the fact that fewer and fewer components are available for through board mounting. A secondary factor is that the project is envisaged to be a long running activity and as such, any components used need to be available long term. Usually, the most effective method of ensuring continuity of supply is to select generic products that have been in the market place for many years. This unfortunately means that the design will potentially require more components than a customised design has and also this impinges on the PCB design layout.

Other significant risk and problem areas of this project will be :-

- Getting the programmer to be recognised by chip manufacturers.
  - (Solution) Initially only devices that have published programming algorithms will be able to be programmed. This still represents a significant number of devices.
- Finding a highly cost effective plug and socket system for sub module docking.
  - (Solution) Initially sub module boards may be hard wired.
- Safety issues of using a programmer without a case. The board is a low voltage device, however, desk contamination could damage the programmer rather than being a particular threat to the user.
  - (Solution) allow legs and protection sheet to be fitted to PCB to give required isolation.
- System reliability due to high component count.
  - (A future solution) route would be to develop an ASIC (Application Specific Integrated Circuit) that could replace the contents of the whole daughter board. However, for this project, the development costs alone of an ASIC, rule it out as a possible solution option.

- On the development front, the main risk areas will be :-
  - Large PCB Layouts (Processor Board and Mother Board).
    - (Solution) Use Auto route and vias (or spend lots of time resolving).
  - Complex Application Software programme implementation.
    - (Solution) Decompose use effective analysis techniques.
  - Development of an Integrated Macro Processor programme.
    - (Solution) Decompose use effective analysis techniques
  - Component sourcing due to standard item NOW only available in surface mount format.
    - (Solution) Migrate to surface mount route when funding available.
4.35 Strategy for risk management.

One of the simplest methods to identify risk areas is to allocate a risk factor quotient to each activity in the project. For purpose of ease, the following method has been used:-
Each task is allocated a potential risk factor from 1 to 5 where 1 is considered to be a low risk task , whereas 5 is considered to be cause for concern.
The following table categorises the guidelines used.

<table>
<thead>
<tr>
<th>Risk Level</th>
<th>Category of Task Risk Level Allocation.</th>
<th>Difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Task expected not to overrun by more than 25%</td>
<td>Easy</td>
</tr>
<tr>
<td>2</td>
<td>Task expected not to overrun by more than 50%</td>
<td>Medium</td>
</tr>
<tr>
<td>3</td>
<td>Task expected not to overrun by more than 100%</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Task expected not to overrun by more than 200%</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Task expected to be a potential major problem</td>
<td>Concerned</td>
</tr>
<tr>
<td>+1</td>
<td>Task expected to have increased likelihood of difficulty.</td>
<td></td>
</tr>
<tr>
<td>+2</td>
<td>Task expected to have significant likelihood of difficulty</td>
<td></td>
</tr>
</tbody>
</table>

Note
Any task that exceeds level 5 needs to be re-evaluated as its outcome may seriously effect the projects viability.

The document Appendix Filename PERT_EIE.XLS shows the Risk Allocation levels and summation averaged calculation. As the average value of all risks is below 3 then it could be expected that the project should succeed within the project constraints.

4.4 Decision Analysis

This section will cover :-
Improving decision making process.
Decision making and its impact on risk.
Decision tree.
Forecasting.
4.41 Improving decision making process.

To make effective decisions, it is necessary to collate ideas, information and resource availability from as many sources as possible. No ideas should be rejected without prior consideration, however, a time limit should be pre-set on the collection procedure. The next phase is to identify where the possible risks associated with the project are and their implications. Having identified the risks and grouped the ideas, it is now important to identify the activities, setting realistic milestones and to outline resource requirements.

Having got a basic plan, the first review will check feasibility and viability of continuing. At this stage, very little costs should have been incurred and abandoning now will normally cause minimal disruption.

If the review indicates that the project is viable (i.e. potentially profitable), then a second review is implemented to clearly identify the following items:

- The project specification.
- The project activity plan.
- The project funding.
- The availability of the required resources.
- The milestones and individuals responsible for them.
- The contingency procedures and insurance.

At this stage, providing the project plan is followed and milestones met, then the project should have a successful outcome.

4.411 Decision making and its impact on risk.

All decisions involve some level of risk. By developing a diagram or model like the decision tree, then the project manager should be able to quickly see the implications of each decision and its potential outcome. Obviously, as the decisions become more complex then more sophisticated decision making tools may be needed. These tools will assist with the decision making and decomposing of the decision options to more manageable levels. As the levels are decomposed, the associated level of risk should also be more easily definable and quantifiable. Once the project manager has been able to quantify the risks associated with any decision, a decision table should easily be constructed. Even relatively arbitrary decision about perceived risk magnitude of a particular activity can cumulatively identify an unacceptable route when the project is viewed as whole.
4.42 Decision tree.

The document Appendix Filename DTREE.DOC shows an example of the decision tree that has effected the direction of the project. It can be seen that as cost is a major constraint, most alternative options have been ruled out as a possible route.

4.421 Forecasting.

Prediction of the future is always a difficult activity as we are dealing with the unknown. However, we can use past performance as a guide to how things may develop provided all things remain the same as in the past. If we are dealing with the unpredictable i.e. like the stock exchange, then we have little to base our decisions on other than intuition and insider information. However, with a project, hopefully we have some experience or can call upon experience to base our decisions on. With past experience and some of the modelling software tools like UML, MS PROJECT and YOURDON, we should be able to make moderately accurate predictions of expected work loads and resource requirements. The GANTT chart document Appendix Project GANTT Chart shows the effect of the single resource being multi-tasked. The time scales have been expanded into September, however, this does show the implications of over allocation of that single resource. It is fortunate that a number of duration estimated have so far been generous or there would be potentially a need to recruit extra resources to complete the project by the deadline.
5.0 Hardware Design.

5.1 Design Plans.

The initial project product design phases focused on the ability to programme a limited range of devices from the PIC processor family. The design was left sufficiently open so that it could easily be upgraded to extend from the initial design processor range to other processor families and logic devices. Typically, these upgrades needed to include features such as :-

- more active pin connections.
- all pins supporting full range programming voltage capability.
- all pins configurable as Power or Ground connections.
- all pins being dynamically configurable as Inputs or Outputs.
- every pin capable of behaving as a master device clock.

The machine’s configuration and programming methods should be implemented through a dynamic structure which would be downloaded from a central store. The central store typically would be an Open Access Public Domain Internet site. The central store of dynamic structures could easily be upgraded as “change request” requirements dictate.

The whole system will be controlled and run from a standard DOS or Windows based Personal Computer. Future consideration would be given to the possibility of running the applications from other platforms i.e. LINUS, Apple Mac. However, the amount of resources that these options would involve put it beyond the scope of this project.

Due to a limited project budget, the programmer would need to be built in various “flavours” i.e. The professional version would have the smarter casing, additional interface options (Serial / Parallel Port plus USB), a stand alone control mode and multiple programme device sockets. The hobbyist (or custom build) version would support the same functionality, however, with none of the frills.

5.2 General Requirements.

The Device programmer will be controlled by a number of constraints. These are :-

- The System has to be low cost, price competitive and market place viable.
- The System has to be highly flexible to cope with past, present and future devices.
- The System life time expectancy of the product is anticipated to be many years.
5.3 Initial Block Diagram.

5.4 Detail Block Diagram.
5.5 Detailed Requirements and Implementation.

This section covers the following areas :-

- The ZIF (Zero Insertion Force) Pin Interface.
- The Voltage generators.
- The Reference Voltage Generator.
- The Voltage Monitors.
- The Digital Interface Section.
- The Process Control Interface.

5.51 The ZIF Pin Interface.

The ZIF Zero Insertion Force) Pin interface has the following requirements :-

- Each Pin must be able to tie a pin to ground.
- Each pin must be able to supply a Voltage.
- Each pin must be able to be an Input.

To enable these features, the following discrete transistors switching circuits were implemented.

**The Ground Pin Tie configuration.**

Whilst the Base of the transistor is tied to ground, the collector is effectively an open circuit with the transistor shut off. However, when the Base is driven with a voltage above +0.6 Volts, then the transistor can start to conduct. When sufficient current is driving the base, this effectively ties the collector to ground.

![Diagram showing the Ground Pin Tie configuration](image-url)
The Voltage supply Pin.

In this configuration, when the Base of the Transistor is tied to ground, then the Emitter is effectively an open circuit with the transistor shut off. However, when the Base is driven with a voltage above +0.6 Volts, then the transistor can start to conduct. With sufficient current driving the transistor, it will effectively appear as a short circuit hence switching a supply voltage. As with the previous circuit, there will be a small voltage difference due to the saturation voltage of the transistors. However, for all practical purposes, it can be ignored as it will typically be only circa 0.2 Volts.

The Basic Control Circuit.
To avoid the necessity of requiring large voltage swings to control the system, the following circuit was developed to enable a simple option to be implemented. A second transistor was added to enable the base of the control transistor to be either tied to Ground or allowed to be pulled to +VOLTS through the Resistor R. While the control voltage is less the +0.6 Volts, the lower transistor will be effectively shut “off” and the control transistor will be driven into saturation. When the control voltage switches the lower transistor fully “on”, then the control transistors base will be held effectively at Ground and the Emitter will appear as a high impedance.
The circuit described so far covers all the basic requirements of the Pin interface requirements, however, refinements were still required. The Diode and Resistor chain were added for the following reasons:

- The Resistor limits the drive current and loading of the Transistor on the external control circuits.
- The Diode is used to isolate the Transistor circuit from the Control circuit.

To give all the required features of the pin interface, the following circuit is built from a combination of the previous circuits. The circuit has the two digital control signals that in combination can present the three distinct required output states.
Safety Aspects.

As a safety feature when “Ground Request” is asserted, it also asserts an overriding forced “TriState” request via the additional Resistor and Diode chain. Therefore, the destructive Ground and Voltage option can never occur even if the request is selected or driven by the programmer.

5.52 The Voltage Generators.

The Voltage generators are of a standard basic design in that the operational amplifiers are configured as a simple multiplying adder circuit. The value input resistors are adjusted in conjunction with the feedback resistor so that each input has a power of two gain higher than its predecessor. The additional Transistor is used to increase the power output capability of the operational Amplifier. To get the required voltage range i.e. Zero to Thirty volts, the operational amplifiers are configured in the single supply mode with the negative rail voltage being used as the common earth line. The resistors were selected to give as close as possible a monotonic output. Only seven of the Digital drive signals are used so as to keep the resistor range with reasonable and usable bounds.
5.53 The Reference Voltage Generator.

The Reference Voltage generator was constructed in a similar manner, however, due to the limited supply current requirements, there was no need for the additional transistor for increase power capability. An alternative method was also implemented using commercial 8 bit DAC to drive the amplifier. A precision 5.6Volt zener diode is used as a reference for the DAC device. The calibration procedure offers the opportunity to adjust the amplifier gain for correcting any minor voltage errors.

![Diagram of Reference Voltage Generator]

5.54 The Voltage Monitors.

A Voltage Monitor compares the actual ZIF pin voltage with the Master reference Voltage. The level shifter network configures the output to a valid TTL Monitor level. The Monitor Level is used for calibration procedures. Any instability of the Monitor Level output is removed by latching the Reference voltage up or down an increment.

![Diagram of Voltage Monitor]
5.55 The Digital Interface.

This interface implements the majority of the programming capability. Each ZIF (Zero Insertion Force) Pin has the following control inputs:

- A Signal that requires a voltage to present on a pin.
- A Signal that ties a pin to ground.
- A Signal that enables the Master Clock 1 to control a voltage present on a pin.
- A Signal that enables the Master Clock 2 to control a voltage present on a pin.

The Master Clock signals 1 and 2 are both TTL levels and are either generated by the Master Clock module or supplied from an external source. The main reason that there are two clock signals is that some devices need to be stimulated with clock signals in anti-phase. There is also a possibility that some processors may need more than one control clocking process to be effectively programmed.

The Digital logic implement all the required functionality with the two output signals which are:

- Set pin to Voltage Ground.
- Voltage select or pin is in TriState mode (input mode).
5.56 The Process Control Interface.

The process control interface is where the majority of the system intelligence resides. The processor will control the following functions:

- Data transfer to and from the Personal Computer (PC).
- All the timing functions of the Programme Process logic via the Parallel Port lines.
- The Implementation of the programming algorithms.
- The Implementation of Device read algorithms.
- The System Diagnostics and Calibration Procedures.
- Manage Flash disk Memory interface.
- Verification of Algorithm data expiry

The Process Control Interface (Basic Version).

A more detailed Block diagram of the Process Control Interface can be located in the Appendix. The present Process Control Interface is based around a Hitachi 64180 processor, however, the basic design of the programming elements ensures that they are fully compatible with any processor that can support a TTL interface.
6.0 Software Design.

The Software design falls into two distinct areas :-
- Software for the Development System.
- Software for the Device Programmer.

6.1 Software for the Development System.

The Software for the Development system consists of the following programmes :-
- The General Purpose Cross Assembler.
- The General Purpose Macro Processor.
- The General Purpose Linker.
- The General Purpose Simulator.

6.11 The General Purpose Cross Assembler.

The basis of this system is that one tool set should be suitable to enable the development of any embedded style processor. When viewing Assemblers in particular, there is a number of characteristics which are noticeably common between all systems. When looking at a typical Assembler programme source, we can identify a number of distinct areas of commonality.

.TITLE Example Programme

.ORG Here+0CH ;Programme Start Address

X1: LOAD (J1),Y2 ;Store Data
     SUB Y2,#1. ;Change Data
     INC J1 ;Change Data Pointer
     IF_ZERO Y2 X1 ;Loop

.END ;End of Programme

The source above can be broken down into the following distinct areas :-
- The “Blue” Text identifies Directives.
- The “Green” Text identifies Comments.
- The “Orange” Text is a Label (A Special Symbol).
- The “Violet” Text is an Expression.
- The “Grey” Text is a Symbol or a Data Store.
Most systems therefore need the following facilities: -

- The ability to process “Directives”.
- Support for “Labels” and “Symbols”.
- An “Expression Evaluator”.
- The ability to specify “Data Stores”.
- An Instruction Set Code Translator.

Basic Data Flow Diagram of the Cross Assembler.

The Basic control Mechanism of the Cross Assembler.
Main operations of the basic features.

The **Control Process** is Broken down into the following passes :-
1. Process Source Directives , Basic Translation of Mnemonics.
2. Resolve Label addresses and Symbols, Substitute Symbol values into Object code.
3. Dump Object or Report errors.

The **Symbol Table Manager** needs to be able to :-
- Initialise the Symbol Table structure.
- Store or Add contents a Symbol.
- Locate or Get contents a Symbol.
- Update a Symbol store contents.
- Evaluate or resolve Symbol contents.
- Display or Dump Symbol table contents.

The **Data Store Manager** needs to be able to :-
- Locate appropriate Data Store access regions.
- Write data to its Data store location.
- Read data from its Data store location.
- Error report bad Data store access.

The **Expression Evaluator** needs to be able to :-
- Translate values using differing radix or formats.
- Perform arithmetic and logic operations on values.
- Access contents symbol in Symbol table.

The **Parameter Processor** needs to be able to :-
- Divide a text line into individual entities using known terminators.
- Remove white space and redundant controls.

The **Directive Processor** needs to be able to :-
- Purge comments and redundant information from a text string.
- Recognise specific text strings.
- Extract any associated parameters with recognised string.
- Perform the appropriate action of detected directive.

The **Language Code Interpreter** needs to be able to :-
- Initialise “Application Specific Language” library pointers.
- Repeat activities till EXIT Encountered.
  - Access next statement.
  - Access command features of “Application Specific Language” switch table.
    - Process selected “Application Specific Language” command.
- End Repeat.
- Exit Language Code Interpreter.
The only item that is specifically unique in all Assemblers is the Instruction Set Code Translator. With this concept in mind, if the translation of the Instruction Set can be processed via some externally linked entity, then all the remaining items could be processed by a single package. The methodology used to resolve this problem was to develop an interpreter that could perform the translation process. The translation interpreter process was developed around the concept of using an “Application Specific Language” for both the “Process” and “The Documentation”.

The General Purpose Cross Assembler Shell process separates the functionality of the system in the following manner:

The General aspects of the Cross Assembler:
- Processes all the Directives.
- Evaluates the Expressions.
- Maintains Symbols in Symbol Table.
- Prepares the Data Stores.
- Creates Parameters from any remaining programme Statements.
- Runs the “Application Specific Language” Interpreter with supplied Parameters.
- Displays and Outputs Processed Results.

The “Application Specific Language”:
- Initialises the Assembler default settings.
- Translates Parameters into a valid Data Store Format. (No Checking needed).
- Adds Appropriate Range checking parameters and data size specifications to Data Store Statements.
- Creates Error messages as appropriate.

Basic Data structure sections of the Cross Assembler.

<table>
<thead>
<tr>
<th>Pointers and Control Parameters</th>
<th>Source Code Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol Table Store Area</td>
<td>Parse Code Area</td>
</tr>
<tr>
<td></td>
<td>Object Code Area</td>
</tr>
</tbody>
</table>
6.12 The General Purpose Macro Processor.

**Basic Data Flow Diagram of the Macro Processor.**

**The Macro Processor**

- Code Source or Sources
- Macro Library Source
- Converted Source
- Macro Process Configuration Options
- User Interface

**The Basic control Mechanism of the Macro Processor.**

- Symbol Table Manager.
- Data Store Manager/s.
- Expression Evaluator.
- Control Process
- Parameter Processor.
- Directive Processor.
Basic Data structure sections of the Macro Processor.

The Macro Processor has an identical data structure to the Cross Assembler, however, some areas are used for slightly different purposes. The majority of the Macro Processor common named entities operate identically as for those in the Cross Assembler, the only exception being the Control Process.

The Control Process is Broken down into the following passes :-

Pass 1.
- Processes the unique operation Directives.
- Extracts any Embedded Macros and Stores them in the Macro Area.

Pass 2. The Recursive section.
- Implement Repeats, Expands Macros,
- Substitutes Parameters, Processes “IF” blocks
  Implemented by :-
  - Read data from Source area.
  - The Processed or modified data is loaded into Workspace area.
  - A tidy process copies the updated data back to Source area.
  - If Recursive count NOT expired and WORK still to do
    Continues till all processes complete.

Pass 3. The Termination Options :-
- Option 1. Output processed Source.
- Option 2. Return control to Cross Assembler.
- Option 3. Quit the Macro Processor.
6.13 The General Purpose Linker.

Basic Data Flow Diagram of the Linker.

The Linker

Programme
Object or
Objects

User
Interface

Programme
Map

Programme
Run Code

Linker
Configuration
Options

The Basic control Mechanism of the Linker.

Symbol Table
Manager.

Data Store
Manager/s.

Expression
Evaluator.

Control
Process

Parameter
Processor.

Output Format
Converter
The Linker has a similar data structure to the Cross Assembler, however, some areas are used for slightly different purposes. The majority of the Linker common named entities operate identically as for those in the Cross Assembler, the main exception being the **Control Process**.

The **Control Process** is Broken down into the following passes :-

Pass 1.
- Remove all redundant information.
- Extracts Object Binary and Convert Address Directives.

Pass 2.
- Tidy Symbol Table.

Pass 3.
- Convert Binary to required Output format.
6.14 The General Purpose Simulator.

Basic Data Flow Diagram of the Simulator.

The Simulator

Run Code Source or Sources

Simulator Library Source.

Simulator Configuration Options

User Interface

Report Log File.

The Basic control Mechanism of the Simulator.

Display and Data Log Manager.

Data Store Manager/s.

Expression Evaluator.

Control Process

Parameter Processor.

Command Line Processor.

Machine Code Simulator.
The Control Process of the Simulator.

The Simulator uses a number of common entities with other packages in the system, however, there is a major processing concept differences with the Control Process:

- The “Command Menu Mode” identifies how many Cycles are to be processed based on the User response. It is only a QUIT response in this mode that will terminate any further the Simulator activities. This section also controls the ability to include executable programme code and to change the status of system data store contents.
- “Perform a Processor operation” activates the Simulator Interpreter to process a Processor Machine code operation.
- “Perform a Device operation” activates the Simulator Interpreter to process a Device activity.
- “Update Data Display Page” redraws the currently selected information screen.

The Data Store Manager/s needs to be able to:

- Create the appropriate Dynamic Data structure and Mapping references.
- Read or Write to an individual Region elements.
- Error report if Region access location invalid.

The Machine Code Simulator Interpreter needs to be able to:

- Initialise “Application Specific Language” library pointers.
- Repeat activities till EXIT Encountered.
  - Access next statement.
  - Access command features of “Application Specific Language” switch table.
  - Process selected “Application Specific Language” command.
- End Repeat.
- Exit Machine Code Simulator Interpreter.
Basic Data structure sections of the Simulator.

Basic Simulator Data Structure.

- Pointers and Control Parameters.
  Direct Access Fixed Size Structure.
- Pointers and Storage Areas.
  Created from Simulator Library.
- Pointers and Storage Areas.
  Created from Configuration File.
- Regions Contents Storage Areas.
- Regions Contents Storage Areas.

Basic Region Mapping Method of the Simulator.

Region Pointers “A”

Region “A” Maps
- Map Definition ‘1’
- Map Definition ‘2’
- Map Definition ‘.’
- Map Definition ‘m’

Region “A”
- Physical Store ‘1’
- Physical Store ‘2’
- Physical Store ‘.’
- Physical Store ‘n’

Note A Map definition may span many Stores, just part of a Store or even Share Stores in Multi Access mode.
6.2 Software for the Device Programmer.

This Software entity is expected to be an ongoing project for a number of years. It is likely to experience a significant number of development extensions before it can be considered to be a stable product.


![Diagram showing the basic data flow of the Device Programmer]

The Basic control Mechanism of the Device Programmer.

![Diagram showing the control mechanisms of the Device Programmer]

- Time Control Manager
- Data Storage Manager
- PIO Interface Driver
- Control Process
- Calibration Manager
- Interface Device Drivers
- System Monitor & Device Manager
7.0 User Interface.

There are two distinct interfaces to the overall system. These are :-
- The Software development route.
- The Hardware or Device programming route.

As these interface are quite significantly different they will be discussed as two separate unique entities.

7.1 The Software Interface.

This section specifically focuses on the user interface programme development environment aspects of the Universal Programming development system.

7.11 The Integrated Development Environment.

The Integrated Development Environment has been developed especially for the new user to the Universal Programming development system. Many new programme developers due to their prior background experience are somewhat unfamiliar with software that is driven by command prompts. This lack of familiarity can be overcome by offering the user an on screen menu display where a limited number of options are presented. The user selects his required option to achieve his required outcome. There is one potential danger with the excessively user friendly interface and that is, many users begin to expect that all application systems are going to tell them what to do. The very nature of developing software for embedded system should imply that the user knows his device and its constraints. The Universal Programming development system cannot and should not be expected to be a teaching application. The user should understand basic requirements and development sequence process before embarking their first programme.

The normal development sequence is :-
- DEFINE the work environment.
- EDIT programme source or CREATE a new programme source.
- ASSEMBLE Source (Back to EDIT if Errors detected).
- LINK appropriate Object (Back to EDIT if Errors detected).
- SIMULATE task image (Back to EDIT if functionality incorrect).
- Submit task image to device programmer.

This mode is perhaps more appropriate for the more experienced programme developer who is working with multiple programme sources. However, continually switching between manual command mode and IDE mode may not be an unreasonable working practice approach, especially as the INIT.EXE programme can set up most of the appropriate default response replies. INIT.EXE is the first programme that is used because it prepares the environment. The rest of the process is as follows :-

- EDIT programme source or CREATE a new programme source.
- ASSEMBLE Source (Back to EDIT if Errors detected).
- LINK appropriate Object (Back to EDIT if Errors detected).
- SIMULATE task image (Back to EDIT if functionality incorrect).
- Submit task image to device programmer.

7.2 The Hardware Interface.

The device programmer is connected to the host system either by :-

- Using a Serial Interface Link.
- Using a USB port.

If the system is connected using a Serial Interface Link when running under WINDOWS 9x or similar system, then one appropriate programme to control the Hardware is the “Hyper Terminal” terminal emulator programme. Once the communication link is established, then the Programmer Hardware will display a menu that will define what functions are available to the user.

Note: As the interface options to the programmer is a universal format, then the control options should be viable on any system that support that interface. The only difficulty may be with the transfer of the data file structures. All communications to and from the programmer will be in an ASCII text format.

Typical operations that should be available are :-

- The option to Download the programming algorithm.
- The option to Download the data to be programmed.
- The option to Upload the data from a programmed device.
- Configuration and Calibration options.
8.0 System testing.

Due to the very large scope and nature of the project a number of different strategies have been implemented. The typical approach for most Software entities was to test sub modules for validity. When the basic functionality was proven, the sub modules were integrated into the complete system where the next level in the test sequence was implemented.

8.1 Hardware Testing Strategy.

The whole product was made from a large number of common entities. The complexity of the system was not in the individual items but in the fact that there is a significant quantity of each item. The complexity is also present in the way the sub entities can interact with each other.

8.11 The Analogue Hardware.

The Analogue Hardware entities in general were initially developed and tested in a simulated mode using the CADENCE evaluation version of PSPICE. Once the Simulated versions gave the required results, then some of the circuits were built up using a standard prototype bread board circuit. When the Bread board prototypes gave proven results, the circuits were offered for inclusion into the main system. The system is a hybrid Analogue and Digital system and it was necessary at times to rely on the prototype printed circuit boards for the final testing stage. It is for this reason that the test procedures have been so extensively developed, however, the test procedures are now ready when and if the product goes into production. See appendix item marked Analogue Simulation Results detailing some of the hybrid circuit design aspects.

8.12 The Digital Hardware.

The Digital Hardware entities are in general initially checked with simple continuity checks verifying that signal and power routing is correct. Power is applied to the board and voltage checks are taken to verify power routing is correct. The remaining checks are normally software driven and selected pins are monitored with an oscilloscope.
8.2 Software Testing Strategy

The testing of the Software entities i.e. The Assembler, The Linker, The Macro Processor, The Dis-Assemblers and The Simulator are usually tested as a group. Each processor configuration has an example source that contains one or more variants of every style of instruction that processor should be able to perform. Usually the test suite has all the instructions sorted in alphabetical order. The test suite is submitted to the Assembler, the source is Linked and then Dis-Assembled. Manual checks are then performed to verify that the Dis-Assembly output matches the original source. A second source is then developed that will also attempt to use an example of all the instructions of a processor, but this time, the source is expected to produce some known expected result. The second source is Assembled, Linked and submitted to the Simulator. The Simulator will process the programme and should give the expected results that the second source was expected to produce. The software is then submitted for usage testing. Whenever faults or anomalies are identified, they are corrected and the new software is released with the next system revision update.

The test procedures for the Programmer Hardware is based very much around a production line testing. The actual test procedures are described in the document TESTPROC.DOC and a copy can be found in the Appendix. The majority of the Hardware checks are controlled by a series of mini test programmes. An example of a mini test programme suite can be seen in the document BOARDTST.ASM also located in the Appendix.
9.0 Conclusion.

9.1 The Successes.
With hindsight, this project was significantly larger than had been initially envisaged. After the final update of the GANTT chart had been made and the resource allocation levelled, the GANTT chart indicated that the project was at least five months ahead of schedule (based on 20 hours being allocated to the project per week).

The Software development system has shown itself to be a proven and viable entity if only by the fact that it has been used exclusively to develop the software for the General Purpose Programmer. The introduction of the Macro Processor into the development suite has significantly enhanced Data and Programme structures options that can be used within the development programme source code. It is the Macro Processor specifically that has enabled the ZMON source to be built and hence permitting it to control the whole of the General Purpose Programmer Test Suites and its Main Application Programme. The Simulator has proven to be an invaluable tool during the system fault finding if only to confirm that the binary image was valid and was functioning in the manner specified. It has also been used as a debugging aid when code did not perform quite as expected.

9.2 The Design Methods.
The Application Specific Language concept has shown by example how a PIC Cross Assembler can be developed in an extremely short time period. This concept shows the total flexibility and efficiency it can offer to a system developer.

The use of Box Concept analysis has also shown itself to be an effective methodology for designing small systems both for the hardware and software aspects of the project.

9.3 The Hardware learning activities.
During the project design phase the PSPICE Simulation tool has been highly useful in evaluating much of the analogue circuitry, however, it does not permit one to ignore the need to build some circuits on a breadboard just to verify results. This was brought sharply into focus when a \( \mu \)741 chip was used instead of a section of an LM348 which was supposed to have identical characteristics.

One of the main lessons learnt during the processor hardware development exercise was that Test Equipment can also go faulty. It was as a result of running the test programmes in the General Purpose Simulator showing them to be viable that the fault finding focus changed. Rather than examining the new hardware, it was considered that the Test equipment may not be performing correctly. The problem was eventually tracked down to be an earth loop fault located on the EPROM Emulator socket connector. The effects of that fault did stall the project for a significant time period as the interactions were of an intermittent and inconsistent nature.
9.4 The Hardware Units.
Although a complete set of boards (some 30 in total) have not been produced, a prototype of every variant has been fabricated. The Test procedures designed for the production phase of the manufacturing process have shown that the design of every board is sound. All the boards perform to specification, however, some minor modifications to them may be implemented to help reduce the manufacturing time. Once sufficient quantities have been manufactured, then the system will be ready to begin the next phase in the process which is the implementation of the device programming algorithms.

9.5 Future Development.
Now that the development prototype modules have proved themselves, a second phase design review needs to take place. This will specifically look at the implementation of using:

- Programmable logic devices to minimise the component count.
- Surface mount components to allow more compact PCB to be designed.
- Alternative interconnection methodology to reduce overall plug and socket costs.
- An appropriate casing to enclose the product.
- Alternative cost effective methods to implement the Analogue circuitry.
- An Alternative cost effective and labour efficient method of producing Printed Circuit Boards.
- The use of the prototype programmer for programming the devices in the next generation of programmer.

The General Purpose Assembler and Simulator now supports a limited environment and programming capability for PIC processor range. However, a fully viable market-ready embedded microprocessor development system for PIC processors will still require considerably more functionality to be added to the system. Specifically, the Simulator Library code needs the implementation of the Watchdog timer and some appropriate devices so that more practical demonstration programmes can be presented.
9.6 Alternative Development Directions.

9.61 FPGA Processing.
In pure concept terms, the General Purpose Programmer accepts a binary image pattern and burns that pattern into a specified device. One type of pattern may be used to implement some logic functionality in a programmable logic device. The next binary image could be some machine code to implement an application in a microprocessor. Only the method of production of the binary image differs. The General Purpose Assembler basically translates mnemonics of a specified processor to binary codes. It may be possible to develop a series of assembler style languages targeted at a specific range of logic devices. The General Purpose Assembler could then create the appropriate binary patterns from a source code necessary to configure the selected logic devices. This may be a future development path for the General Purpose Assembler.

By connecting some of the programming pins of the General Purpose Programmer via discrete resistors to previously allocated connections on the ZIF socket, it should be possible to use the programmer to analyse the functionality of an unknown device by stimulation of its pins. Once the programmer has captured a signature from the unknown device, it could compare that signature against known templates for part code recognition.

9.63 Device Emulation.
The General Purpose Programmer can derive logic signals to a pin and read levels present on that pin. Therefore, with a suitable adapter, the programmer could become a General Purpose Device Emulator. Obviously, the operational speed of the programmer is currently relatively slow, however, with some design changes a better performance should be realisable. Hence the programmer could not only program devices but also emulate them in real time prior to the burning process. This could be a highly cost effective option for future system developers.
List of Useful References.

<table>
<thead>
<tr>
<th>Title of Document</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>HD64180 Hardware User Manual. 1987</td>
<td>UM64180/88/03 Hitachi Ltd</td>
</tr>
<tr>
<td>PIC16C84 Data Sheet</td>
<td>DS30445A ©1996 Microchip Technology Inc.</td>
</tr>
<tr>
<td>PIC16F87X Data Sheet</td>
<td>DS3029C ©2001 Microchip Technology Inc.</td>
</tr>
<tr>
<td>Programmable Logic Devices 1986</td>
<td>Mulard Publication no. 806352</td>
</tr>
<tr>
<td>PAL Device Handbook AMD and MMI</td>
<td>1988 Advanced Micro Devices Inc. 10173A/0</td>
</tr>
<tr>
<td>PAL Device Handbook AMD and MMI</td>
<td>1988 Advanced Micro Devices Inc. 10206A/0</td>
</tr>
<tr>
<td>The TTL Data Book for design Engineers 1984</td>
<td>Vol. 1 , ISBN 3-88078-037-4</td>
</tr>
<tr>
<td>The TTL Data Book for design Engineers.</td>
<td>Vol. 2 , ISBN 3-88078-042-0</td>
</tr>
<tr>
<td>TI High Speed CMOS Logic Data Book 1988</td>
<td>ISBN 3-88078-071-4</td>
</tr>
<tr>
<td>8 Bit Multiplying Digital to Analogue Converters</td>
<td>SLAS061C Revised November 1998</td>
</tr>
<tr>
<td>TLC7524C, TLC7524E, TLC7524I</td>
<td>Texas Instruments</td>
</tr>
</tbody>
</table>
APPENDIX

Section

This section consists of the following pages

<table>
<thead>
<tr>
<th>Title of Pages</th>
<th>Number of Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Diagram of General Purpose Programmer.</td>
<td>1</td>
</tr>
<tr>
<td>Block Diagram of General Purpose Microprocessor Module.</td>
<td>1</td>
</tr>
<tr>
<td>Programmer Interface Block Diagram.</td>
<td>1</td>
</tr>
<tr>
<td>The Project GANTT Chart</td>
<td>3</td>
</tr>
<tr>
<td>The Project PERT Chart.</td>
<td>3</td>
</tr>
<tr>
<td>Activity calculation spreadsheet Filename PERT_EIE.XLS.</td>
<td>1</td>
</tr>
<tr>
<td>Work Analysis Calculations Filename PERTCALC.doc</td>
<td>2</td>
</tr>
<tr>
<td>Decision Tree Diagram DTREE.DOC</td>
<td>1</td>
</tr>
<tr>
<td>Analogue Simulation Results.</td>
<td>A Sub-Section</td>
</tr>
<tr>
<td>Circuit Specifics of the General Purpose Programmer.</td>
<td>A Sub-Section</td>
</tr>
<tr>
<td>Documentation, Demonstration and Test Programmes.</td>
<td>A Sub-Section</td>
</tr>
<tr>
<td>Application Specific Language Source code for PIC16.</td>
<td>A Sub-Section</td>
</tr>
<tr>
<td>Application Specific Language Source code for PIC16SIM.</td>
<td>A Sub-Section</td>
</tr>
</tbody>
</table>
Block Diagram of the General Purpose Device Programmer.
Micro-Processor Based on HD64180 Controller.

- 32K Bytes Static EPROM 62256
- 32K Bytes Static RAM 27256
- XTAL for Processor Clock 12.288MHz
- (Optional) FLASH Memory interface
- Optional Onboard Flash Memory Disk Data Store.

- IBM PC Based Application Interface.
  - Main Serial RS232 Interface MAX232
  - USB Converter Interface
  - Optional USB Communications Interface

- Secondary Serial RS232 Interface MAX232
  - Primary Serial Line Communications Interface

- Secondary I/O Parallel Interface 8255
- Primary I/O Parallel Interface 8255

- Keyboard & LED Display Diagnostic Interface 8279

- Data Signals
- Control Signals
- Address Signals
Programmer Interface Block Diagram.

- **Address Interface**
  - LS Bit of Address Buffer
  - Board or Block Selector

- **Data Interface**
  - ADC Board (1)
  - ADC Board (n)
  - ADC Board (Ref)
  - Clock Generator (1)
  - Clock Generator (2)

- **Monitor Interface**
  - Pin Board (1)
  - Pin Board (2)
  - Pin Board (n)
  - Single Bit Tristate
  - 2 Bits Asynchronous

- **Analog Voltages 0-30v**
  - 3 Bits
  - 5 Bit Decode
  - 8 Bits
  - 4 Bits
  - 8 Bits
  - 8 Bits
<table>
<thead>
<tr>
<th>Name</th>
<th>Critical</th>
<th>Critical Milestone</th>
<th>Critical Summary</th>
<th>Critical Subproject</th>
<th>Critical Marked</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID Duration</td>
<td>Critical</td>
<td>Critical Milestone</td>
<td>Critical Summary</td>
<td>Critical Subproject</td>
<td>Critical Marked</td>
</tr>
<tr>
<td>Start Finish</td>
<td>Noncritical Milestone</td>
<td>Noncritical Summary</td>
<td>Noncritical Subproject</td>
<td>Noncritical Marked</td>
<td>Noncritical Marked</td>
</tr>
</tbody>
</table>

Project: General Purpose Programmer
Date: Tue 23/08/11
<table>
<thead>
<tr>
<th>Number</th>
<th>Activity</th>
<th>Optimistic</th>
<th>Most Likely</th>
<th>Pessimistic</th>
<th>Estimate</th>
<th>Variance</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Assembler, Simulator Ready</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Design Assembler ALB</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>3.17</td>
<td>0.25</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Design Simulator Proc SLB</td>
<td>7</td>
<td>12</td>
<td>30</td>
<td>14.17</td>
<td>14.69</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>Design Simulator Dev SLB</td>
<td>9</td>
<td>16</td>
<td>30</td>
<td>17.17</td>
<td>12.25</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>Document ALB SLB</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>5.17</td>
<td>0.25</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>Upgrade Development system</td>
<td>4</td>
<td>20</td>
<td>30</td>
<td>19.00</td>
<td>18.78</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>Design Software IDE Menus</td>
<td>2</td>
<td>5</td>
<td>9</td>
<td>5.17</td>
<td>1.36</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>Produce Demonstration</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4.00</td>
<td>0.11</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>Define MACRO Language</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>4.83</td>
<td>0.25</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>Design MACRO Processor</td>
<td>2</td>
<td>10</td>
<td>25</td>
<td>11.17</td>
<td>14.69</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>Code Language Features</td>
<td>2</td>
<td>7</td>
<td>15</td>
<td>7.50</td>
<td>4.69</td>
<td>4</td>
</tr>
<tr>
<td>14</td>
<td>Integrate into Assembler</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>4.33</td>
<td>1.00</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>Design MACRO Test Strategy</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2.83</td>
<td>0.25</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>Test MACRO Processor</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>4.67</td>
<td>0.44</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>Test ASM Integration</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2.00</td>
<td>0.11</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>Document Macro Processor</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>4.50</td>
<td>0.69</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>Document Hardware</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>10.00</td>
<td>0.44</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>Design Interconnects</td>
<td>7</td>
<td>8</td>
<td>18</td>
<td>9.50</td>
<td>3.36</td>
<td>3</td>
</tr>
<tr>
<td>23</td>
<td>Design Master Processor</td>
<td>8</td>
<td>10</td>
<td>20</td>
<td>11.33</td>
<td>4.00</td>
<td>3</td>
</tr>
<tr>
<td>24</td>
<td>PCB Layout Processor</td>
<td>9</td>
<td>10</td>
<td>30</td>
<td>13.17</td>
<td>12.25</td>
<td>5</td>
</tr>
<tr>
<td>25</td>
<td>Build PCB &amp; Drill out</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3.00</td>
<td>0.11</td>
<td>2</td>
</tr>
<tr>
<td>26</td>
<td>Populate PCB</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3.00</td>
<td>0.11</td>
<td>1</td>
</tr>
<tr>
<td>27</td>
<td>Test PCB &amp; Processor</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>5.33</td>
<td>0.44</td>
<td>3</td>
</tr>
<tr>
<td>29</td>
<td>Identify Requirements</td>
<td>7</td>
<td>12</td>
<td>20</td>
<td>12.50</td>
<td>4.69</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>Design Software</td>
<td>11</td>
<td>20</td>
<td>30</td>
<td>20.17</td>
<td>10.03</td>
<td>3</td>
</tr>
<tr>
<td>31</td>
<td>Code Software</td>
<td>15</td>
<td>30</td>
<td>70</td>
<td>34.17</td>
<td>84.03</td>
<td>5</td>
</tr>
<tr>
<td>32</td>
<td>Design Test Strategy</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>5.33</td>
<td>1.00</td>
<td>3</td>
</tr>
<tr>
<td>33</td>
<td>Test Software</td>
<td>7</td>
<td>10</td>
<td>18</td>
<td>10.83</td>
<td>3.36</td>
<td>3</td>
</tr>
<tr>
<td>34</td>
<td>System Integration</td>
<td>8</td>
<td>10</td>
<td>20</td>
<td>11.33</td>
<td>4.00</td>
<td>4</td>
</tr>
<tr>
<td>36</td>
<td>Design MotherBoard</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>15.00</td>
<td>2.78</td>
<td>4</td>
</tr>
<tr>
<td>37</td>
<td>PCB Layout Motherboard</td>
<td>7</td>
<td>8</td>
<td>25</td>
<td>10.67</td>
<td>9.00</td>
<td>5</td>
</tr>
<tr>
<td>38</td>
<td>Build Mother Board PCB</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4.00</td>
<td>0.11</td>
<td>2</td>
</tr>
<tr>
<td>40</td>
<td>Design Daughter Boards</td>
<td>7</td>
<td>10</td>
<td>14</td>
<td>10.17</td>
<td>1.36</td>
<td>2</td>
</tr>
<tr>
<td>41</td>
<td>Design/Select DACs</td>
<td>10</td>
<td>20</td>
<td>22</td>
<td>18.67</td>
<td>4.00</td>
<td>3</td>
</tr>
<tr>
<td>42</td>
<td>Simulate DAC Design</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>10.00</td>
<td>0.44</td>
<td>2</td>
</tr>
<tr>
<td>43</td>
<td>PCB Layout Daughter Boards</td>
<td>7</td>
<td>8</td>
<td>12</td>
<td>8.50</td>
<td>0.69</td>
<td>3</td>
</tr>
<tr>
<td>44</td>
<td>Build Daughter Prototype</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3.00</td>
<td>0.11</td>
<td>2</td>
</tr>
<tr>
<td>45</td>
<td>Test Daughter Prototype</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>2.50</td>
<td>0.25</td>
<td>3</td>
</tr>
<tr>
<td>46</td>
<td>Order Prototype Components</td>
<td>2</td>
<td>5</td>
<td>7</td>
<td>4.83</td>
<td>0.69</td>
<td>2</td>
</tr>
<tr>
<td>47</td>
<td>Order System Components</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>3.83</td>
<td>0.25</td>
<td>2</td>
</tr>
<tr>
<td>48</td>
<td>Build Daughter Board PCBs</td>
<td>7</td>
<td>8</td>
<td>12</td>
<td>8.50</td>
<td>0.69</td>
<td>2</td>
</tr>
<tr>
<td>49</td>
<td>Design Pin Voltage Sensors</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>2.67</td>
<td>0.44</td>
<td>3</td>
</tr>
<tr>
<td>51</td>
<td>Produce Feasibility Study</td>
<td>6</td>
<td>7</td>
<td>12</td>
<td>7.67</td>
<td>1.00</td>
<td>3</td>
</tr>
<tr>
<td>52</td>
<td>Produce Feasibility report</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>5.33</td>
<td>0.44</td>
<td>3</td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td>216.00</td>
<td>354.00</td>
<td>622.00</td>
<td>375.67</td>
<td>219.94</td>
<td>112.00</td>
</tr>
<tr>
<td>Averages</td>
<td></td>
<td>5.11</td>
<td>2.80</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overall using Totals</td>
<td></td>
<td>9450.78</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Work Analysis Calculations and PERT.
Program Evaluation and Review Technique.

**Calculations of estimated times and variances.**

The document Appendix Filename PERT_EIE.XLS shows the calculations of the estimated times and the variances for the project.

The estimated times are calculated from the following formula

\[
\frac{OptimisticTime + (4 \times MostLikelyTime) + PessimisticTime}{6}
\]

The variance is calculated from the following formula

\[
\frac{(PessimisticTime - OptimisticTime)^2}{36}
\]

**Layered PERT Diagram.**

The document Appendix PERT Diagram is based on the Appendix Project GANNT Chart and has been organised to show the linkage between task activities. The all red boxes show the sections that have specifically identified to be critical path. Note at this level “all resource overload” had not been fully resolved.

**Estimated time for completion.**

The document Appendix Filename PERT_EIE.XLS shows the summation calculation of the expected completion times.
Calculations of probabilities of project overrun.

\[ \text{variance} = \sigma^2 = \frac{(Pessimistic\,Time - Optimistic\,Time)^2}{36} \]

The following formulae can be used for the calculation of the variance does appear to ensure that the calculated “deviate value z” will also lie in the bounds of standard deviation table.

\[ \text{variance} = \sigma^2 = \frac{\left(\sum Pessimistic\,Times - \sum Optimistic\,Times\right)^2}{36} \]

From the spreadsheet, we can see that the most likely completion time is 354 days.

The probability of 10%, 20% and 50% overruns are calculated as follows:

- the 10% overrun period is 354 + 10% = 389.4 days
- the 20% overrun period is 354 + 20% = 424.8 days
- the 50% overrun period is 354 + 50% = 531 days

\[ \text{normal deviate } z = \frac{T - \text{mean}(\mu)}{\sqrt{\sigma^2}} \]

where \( \sigma^2 = 9450.78 \)

normal deviate for 10% = \( \frac{389.4 - 354}{97.215} = 0.36414 \) the tables give a value of 0.1406
so probability of 10% overrun is = \( (1-(.5+0.1406))*100\% =35.94\% \)

normal deviate for 20% = \( \frac{424.8 - 354}{97.215} = 0.7282 \) the tables give a value of 0.2673
so probability of 20% overrun is = \( (1-(.5+0.2673))*100\% =23.27\% \)

normal deviate for 50% = \( \frac{531 - 354}{97.215} = 1.8207 \) the tables give a value of 0.4656
so probability of 50% overrun is = \( (1-(.5+0.4656))*100\% =3.44\% \)
Decision Tree for General Purpose Programmer Project

Symbols and Legends

- **Return**
- **Outgoing**
- **Income**
- **Probability**

**Choice**

- **Out come**

- **Develop an ASIC**
  - Choice
    - Technology to use
      - Discrete Components
      - Surface Mount PCB

- **Outcome**
  - Too Expensive to fund
    - £180
    - £90
    - £200+£0
    - 0.9

- **Customers order's product**
  - £10
  - £0
  - £-10+£180
  - 0

- **Customer Orders 0**
  - £0
  - £45
  - £0
  - 0.5

- **Customer Orders 2**
  - £200
  - £45
  - £200 + £0
  - 0.5

- **Customer Orders 0**
  - £0
  - £45
  - £0
  - 0.5

- **Customer Orders 2**
  - £200
  - £45
  - £200 + £0
  - 0.5
APPENDIX
Sub-Section

This section contains the following items:

Analogue Simulation Results.

Description:
This is the Analogue Simulation Results section. The section contains specifically the following items:

<table>
<thead>
<tr>
<th>Title of Pages and additional notes</th>
<th>Number of Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>The initial design Analogue Voltage Source Conditioning Selector. Circuit basis assumed discrete programming and chip supply voltages available.</td>
<td>2</td>
</tr>
<tr>
<td>The evaluation of final circuit in switching mode, 2 Variable Small selection option.</td>
<td>2</td>
</tr>
<tr>
<td>Pin Monitor and calibration circuit.</td>
<td>2</td>
</tr>
<tr>
<td>Basic Configuration Level sensor with Pulse Switch feature.</td>
<td>2</td>
</tr>
<tr>
<td>Single Supply 7 Bit DAC. This is an abnormal operating mode for this type of device.</td>
<td>2</td>
</tr>
<tr>
<td>Combined D/A Converter with Pulse switching Circuit. This checked the system performance and loading when applying a high voltage programming pulse to simulated device.</td>
<td>2</td>
</tr>
<tr>
<td>OP Amp Buffer Design for DAC. It was necessary for the circuit to act both as a Buffer and multiply the voltage of the Reference DAC to give the required range. The DAC had to be isolated from feedback voltages higher than +15 Volts. The OP Amp needed to operate in Single Supply Mode to give the required 0 to 30 Volt swing.</td>
<td>2</td>
</tr>
</tbody>
</table>
VOFF = 12.1
VAMPL = 5
FREQ = 333k

R3
D3
Select
220 lF

Q4
Q2N2222

R12
1k

Q5
Q2N2222

R13
1k

Q6
Q2N2222

Ri
200

Ci
20pF

0

Title
2 Variable Small selection option

Size
A

Document Number
Roger Spriggs

Rev
1

Date
Tuesday, July 03, 2001
Sheet 1 of 1
OFFTIME = 100us
ONTIME = 100us
DELAY =
STARTVAL = 0
OFFPV = 1
** Profile: "SCHEMATIC1-Levels"  [ E:\SCHOOL7\Dups\8adplus-schematic1-levels.sim ]
Date/Time run: 09/13/11 15:50:30  Temperature: 27.0

(A) 8adplus-SCHEMATIC1-Levels.dat (active)
This section contains the following items:

Circuit Specifics of the General Purpose Programmer.

Description:

This section contains Circuit and Layout diagrams or Calculations used for the development of the General Purpose Programmer and consists of the following items:

<table>
<thead>
<tr>
<th>Title of Pages</th>
<th>Number of Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circuit Diagrams</td>
<td></td>
</tr>
<tr>
<td>The Micro Processor Board.</td>
<td>1</td>
</tr>
<tr>
<td>The Keyboard Display Controller.</td>
<td>1</td>
</tr>
<tr>
<td>The Main Interface Board.</td>
<td>3</td>
</tr>
<tr>
<td>The ADC(b) Main and Slave Connection Daughter Boards</td>
<td>2</td>
</tr>
<tr>
<td>The Octal ZIF Pin Basic Control Daughter Board</td>
<td>3</td>
</tr>
<tr>
<td>The ZIF Connector Interface Section</td>
<td>1</td>
</tr>
<tr>
<td>Calculations.</td>
<td></td>
</tr>
<tr>
<td>Resistor Division Ratios (RESISTOR.XLS).</td>
<td>1</td>
</tr>
<tr>
<td>Printed Circuit Board Layouts.</td>
<td></td>
</tr>
<tr>
<td>The Micro Processor Board</td>
<td>1</td>
</tr>
<tr>
<td>The Main Interface Board.</td>
<td>1</td>
</tr>
<tr>
<td>The Octal ZIF Pin Basic Control Daughter Board</td>
<td>1</td>
</tr>
<tr>
<td>The ADC(b) Main Daughter Board.</td>
<td>1</td>
</tr>
<tr>
<td>The ADC(b) Slave Connection Daughter Boards</td>
<td>1</td>
</tr>
<tr>
<td>The ZIF Connector Board</td>
<td>1</td>
</tr>
</tbody>
</table>
General Programmer Main Interface
Processor Interface and Control Logic
Author R. J. Spriggs 23/Apr/2002 Rev 1.2
Digital Interface Section
Octal ZIF Pin Basic Control Daughter Board
Author R. J. Spriggs  22/Dec/2001  Rev 2.1
Part Two, Quad Section pins 5 to 8
Octal ZIF Pin Basic Control Daughter Board
Author R. J. Spriggs  22/Dec/2001  Rev 2.1
### Resistor Division Ratios

<table>
<thead>
<tr>
<th>E12</th>
<th>10</th>
<th>12</th>
<th>15</th>
<th>18</th>
<th>22</th>
<th>27</th>
<th>33</th>
<th>39</th>
<th>47</th>
<th>56</th>
<th>68</th>
<th>82</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Div 2</td>
<td>5</td>
<td>6</td>
<td>7.5</td>
<td>9</td>
<td>11</td>
<td>13.5</td>
<td>16.5</td>
<td>19.5</td>
<td>23.5</td>
<td>28</td>
<td>34</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>Div 4</td>
<td>2.5</td>
<td>3</td>
<td>3.75</td>
<td>4.5</td>
<td>5.5</td>
<td>6.75</td>
<td>8.25</td>
<td>9.75</td>
<td>11.75</td>
<td>14</td>
<td>17</td>
<td>20.5</td>
<td></td>
</tr>
<tr>
<td>Div 8</td>
<td>1.25</td>
<td>1.5</td>
<td>1.875</td>
<td>2.25</td>
<td>2.75</td>
<td>3.375</td>
<td>4.125</td>
<td>4.875</td>
<td>5.875</td>
<td>7</td>
<td>8.5</td>
<td>10.25</td>
<td></td>
</tr>
<tr>
<td>Scale Error%</td>
<td>4.166667</td>
<td>0</td>
<td>4.166667</td>
<td>2.272727</td>
<td>1.851852</td>
<td>2.272727</td>
<td>5.769231</td>
<td>3.723404</td>
<td>4.910714</td>
<td>2.941176</td>
<td>3.658537</td>
<td>2.5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E24 Adds</th>
<th>11</th>
<th>13</th>
<th>16</th>
<th>20</th>
<th>24</th>
<th>30</th>
<th>36</th>
<th>43</th>
<th>51</th>
<th>62</th>
<th>75</th>
<th>91</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>Div 2</td>
<td>5.5</td>
<td>6.5</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td>21.5</td>
<td>25.5</td>
<td>31</td>
<td>37.5</td>
<td>45.5</td>
<td></td>
</tr>
<tr>
<td>Div 4</td>
<td>2.75</td>
<td>3.25</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7.5</td>
<td>9</td>
<td>10.75</td>
<td>12.75</td>
<td>15.5</td>
<td>18.75</td>
<td>22.75</td>
<td></td>
</tr>
<tr>
<td>Div 8</td>
<td>1.375</td>
<td>1.625</td>
<td>2</td>
<td>2.5</td>
<td>3</td>
<td>3.75</td>
<td>4.5</td>
<td>5.375</td>
<td>6.375</td>
<td>7.75</td>
<td>9.375</td>
<td>11.375</td>
<td></td>
</tr>
<tr>
<td>Scale Error%</td>
<td>5.769231</td>
<td>1.5625</td>
<td>0</td>
<td>4.166667</td>
<td>0</td>
<td>4.166667</td>
<td>4.651163</td>
<td>5.392157</td>
<td>2.822581</td>
<td>3.333333</td>
<td>3.021978</td>
<td>2.477477</td>
<td></td>
</tr>
</tbody>
</table>

- **Note:** The 1.875 Resistor can be built from two preferred Resistors with values of 1500 + 390 = 1K89 giving a close approximation.
- **Note:** If any Data Bit drives a Resistor value that is noticeably high then a kink occurs. However this can be removed by ignoring the overlapping values and selecting a preferred subset of the values.
  - i.e. If only 64 values are selected this would still give a resolution of ½ a volt per bit when selecting the best patterns during the calibration process.
- **Note:** The Scale Error % compares the division by 8 effect and the next preferred times 10 value from the Resistor Set series of values.
- **Note:** The selected Resistors bands would be the 12 and 15 sets giving the values 12, 6, 3, (1.5 or 15, 7.5, 3.75, 1.875). The final circuit values would be 120K, 60K, 30K, 15K, 7K5, 3K75 and 1K89.
- **Note:** The last resistor is built from two resistors in series, the other values are created by parallel combinations of the main values.
APPENDIX
Sub-Section

This section contains the following items:

Documentation, Demonstration and Test Programmes.

Description:
This section contains documentation and programmes used by the development system and consists of the following items:-

<table>
<thead>
<tr>
<th>Title of Pages</th>
<th>Number of Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Documentation Files.</strong></td>
<td></td>
</tr>
<tr>
<td>General Purpose Programmer Test Procedures.</td>
<td>6</td>
</tr>
<tr>
<td>Filename TESTPROC.DOC</td>
<td></td>
</tr>
<tr>
<td><strong>Test Programmes.</strong></td>
<td></td>
</tr>
<tr>
<td>The Compiled Test Programme listing BOARDTST.LST</td>
<td>14</td>
</tr>
<tr>
<td><strong>Demonstration Programmes.</strong></td>
<td></td>
</tr>
<tr>
<td>Z80TST.ASM a Zilog Z80 programme that uses an 8255 device and compatible with both the General Purpose Simulator and the Z80 based Flight Boards.</td>
<td>3</td>
</tr>
<tr>
<td>8031TST.ASM an Intel 8032 processor programme that uses an 8255 device and compatible with both the General Purpose Simulator and the 8032 based Flight Boards.</td>
<td>3</td>
</tr>
<tr>
<td>PIC16TST.ASM a MicroChip 16C84 processor programme that just tests out some of the instruction set. Can be run using the General Purpose Simulator.</td>
<td>3</td>
</tr>
</tbody>
</table>
Construction and Test Procedures for the General Purpose Device Programmer.

Common Construction and Board Test Procedures.

1. Visual inspection for shorts between pads, vias and tracks. (Clear Problems)
2. Meter Check for shorts adjacent pads, vias and tracks. (Clear Problems)
3. Populate vias, sockets, connectors and any passives.
4. Meter Check for shorts adjacent pads, vias and tracks. (Clear Problems)
5. Check Power rails for correct distribution. (Resolve problems)
6. Apply power in controlled manner. (Check for and resolve problems)
7. Populate active components (As Appropriate {see Specific test procedures}).
8. Apply power in controlled manner. (Check for, and resolve problems)
9. Power Down and prepare for board specific tests.

Test Programmes for Device programmer.

<table>
<thead>
<tr>
<th>Code Name</th>
<th>Start Address</th>
<th>Comment and Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZIF01</td>
<td>1000H</td>
<td>Basic Board Driver &amp; Continuity Test</td>
</tr>
<tr>
<td>ZIF02</td>
<td>1200H</td>
<td>Test Functionality of U21, U22, U23, U24</td>
</tr>
<tr>
<td>ZIF03</td>
<td>1400H</td>
<td>Full Board test</td>
</tr>
<tr>
<td>ADC01</td>
<td>1800H</td>
<td>Basic Board Driver &amp; Continuity Test</td>
</tr>
<tr>
<td>ADC02</td>
<td>1900H</td>
<td>Full Board test</td>
</tr>
<tr>
<td>IF01</td>
<td>2000H</td>
<td>Bare board checks</td>
</tr>
<tr>
<td>IF02</td>
<td>2200H</td>
<td>U12, U13, U14, U15, U17, U18 Checks</td>
</tr>
<tr>
<td>IF03</td>
<td>2300H</td>
<td>U16 &amp; U19 Check Operations</td>
</tr>
<tr>
<td>IF04</td>
<td>2400H</td>
<td>Up, Stop, Down, Stop Ramps from DAC</td>
</tr>
<tr>
<td>IF05</td>
<td>2500H</td>
<td>DAC = Zero/Quarter/Half and Max Volt</td>
</tr>
<tr>
<td>IF06</td>
<td>2600H</td>
<td>DAC Single Bit Steps</td>
</tr>
</tbody>
</table>
The Processor Board (Specific).

If any parts of test procedure fails rectify problem then restart full test sequence.
If EPROM Emulator is being used it will be neccessary to change processor XTAL to 6.144MHz.
Test equipment required is a DVM ,Dual beam oscilloscope and Eprom Emulator.

1. Install U7 and apply +5V to Power board.
2. Verify that U7 Pin 3 = Logic 0 and Pins 6,8 = Logic 1. Press Reset Key verify Pins 3,6 Toggle. Press NMI Key verify Pin 8 Toggles.
3. Using 10K resistor Link Pin 12 to Pin 14 then Pin 12 to Pin 7 verify the Halt LED operates.
4. Power Down and prepare for next test if operations OK.

1. Install U1, U6 and EPROM/Emulator in U3 then apply +5V to Power board.
2. Run Test programme Z80TST01
3. Verify that after Pressing Reset Key the Halt lamp comes on after about 2 seconds.
4. Power Down and prepare for next test if operations OK.

1. Install U4 and EPROM/Emulator in U3 then apply +5V to Power board.
2. Run Test programme Z80TST02
3. Verify that counting patten appears on J1, J2 and J3.
4. Power Down and prepare for next test if operations OK.

1. Apply +5V to Power board. Run Test programme HEX
2. Verify that Full EPROM checks OK by attaching LED board to Port Lines.
3. Power Down and prepare for next test if operations OK.

1. Install U3, apply +5V to Power board, Run Test programme Z80TST03
2. Verify that Full RAM checks OK by attaching LED board to Port Lines.
3. Power Down and prepare for next test if operations OK.

1. Attach Keyboard/Display to J5 apply +5V to Power board, Run Test programme Z80TST04
2. Verify that Display checks OK, Press KEYS and monitor code appear on Port Lines.
3. Power Down and prepare for next test if operations OK.

1. Attach Serial Line Monitor set to 9600 Baud, 8 Bits, 1 Stop Bit, No Parity to either J51 or J52 apply +5V to Power board, Run Test programme Z80TST06 in Output mode.
2. Verify that Monitor Display shows lines of “ABCDEFGHIJKLMNOPQRSTUVWXYZ”.
3. Run Test programme Z80TST06 in Duplex Mode verify input echo mode operational.
4. Power Down and prepare for next test if operations OK.
5. All Test complete, Board now ready for service.
Construction and Test Procedures for the General Purpose Device Programmer.

Octal ZIF Pin Basic Control Board (Specific).

If any parts of test procedure fails rectify problem then restart full test sequence.

1. Connect ZIF Board to Processor Board using (Test lead 10/2*5 Port B LS-5 , Port C MS-5)
2. Run Test programme ZIF01 in Image BOARDTST.
3. Verify that counting pattern appears on U20-Pins 2,3,4,5,6,7 and pulses are also present on Pin,13 on U21, U22, U23 and U24.
4. Power Down and prepare for next test if operations OK.

1. Install U20 and apply +5V to Power board.
2. Run Test programme ZIF01 in Image BOARDTST.
3. Verify that counting pattern appears on U20-Pins 2,3,4,5,6,7 + 18,17,16,15,14,13 and pulses are also present on Pin,13 on U21, U22, U23 and U24.
4. Power Down and prepare for next test if operations OK.

1. Install U21 , U22 , U23 , U24 and apply +5V to Power board.
2. Run Test programme ZIF02 in Image BOARDTST.
3. Verify that walking 1’s pattern appears on U21,U22,U23,U24-Pins 4,5,6,7,8,9,10,11,12.
4. Power Down and prepare for next test if operations OK.

1. Install U10 , U11 , U12 , U13 and apply +5V to Power board.
2. Run Test programme ZIF03 in Image BOARDTST.
3. Verify that Negative going pulse group appear on U10,U11,U12,U13-Pins 3,6,8,11.
4. Power Down and prepare for next test if operations OK.

1. Install U15 , U16 , U17 , U18 and apply +5V to Power board.
2. Run Test programme ZIF03 in Image BOARDTST.
3. Verify that Negative going pulse group appear on U15,U16,U17,U18-Pins 6,12.
4. Power Down and prepare for next test if operations OK.

1. Apply Voltage to (Programme Voltage) VP pins and apply +5V to Power board.
2. Attach ZIF pin Conditioner to ZIF Pins. Run Test programme ZIF03 in Image BOARDTST.
3. Verify that ZIF pins give a similar style presentation to those shown in diagram.
4. Power Down and prepare for next test if operations OK.

1. Install U1,U2,U3,U4 and apply +5V,+25V to Power board.
2. Set Vref to +3V.Run Test programme ZIF03 in Image BOARDTST.
3. Verify that Negative going pulse group appear on U2,U4-Pins 1,7,8,14.
4. Verify that Negative going pulse group appear on U1,U3-Pins 3,6,8,11.
5. Power Down and prepare for next test if operations OK.
6. All Test complete, Board now ready for service.
Construction and Test Procedures for the General Purpose Device Programmer.

ADC Main Daughter Board (Specific).

If any parts of test procedure fails rectify problem then restart full test sequence.

1. Connect ADC Board to Processor Board using (Test leads Port B 10/10 {Note the Special Cable Interceptor must be used to remove +VCC from Pin 10} & Port C 10/6)
2. Run Test programme ADC01 in Image BOARDTST. (All ICs but U3, U4 must be removed)
3. Verify that counting pattern appears on U4-Pins 2, 3, 4, 5, 6, 7, 8, 9, and U3-Pins 1, 2, 3, 4 and a pulse appears on U12, U13, U14, U15 Pin 11.
4. Power Down and prepare for next test if operations OK.

1. Install U3, U4 and apply +5V to Power board.
2. Run Test programme ADC01 in Image BOARDTST.
3. Verify that counting pattern appears on {U4-Pins 2, 3, 4, 5, 6, 7, 8, 9 + 18, 17, 16, 15, 14, 13, 12, 11} {U11, U21, U31 & U41-Pins 2, 3, 4, 5, 6, 7, 8, 9 } also U3-Pins 1, 2, 3, 4 and a walking pattern on U3-Pins 7, 9, 10, 11, 12, 13, 14, 15.
4. Power Down and prepare for next test if operations OK.

1. Install U11, U21, U31, U41 and apply +5V to Power board.
2. Run Test programme ADC01 in Image BOARDTST.
3. Verify that counting pattern appears on {U11, U21, U31, U41-Pins 19, 18, 17, 16, 15, 14, 13, 12} {U12, U22, U32 & U42-Pins 2, 3, 4, 5, 6, 7, 8, 9}.
4. Power Down and prepare for next test if operations OK.

1. Install U12, U22, U32, U42 and apply +5V to Power board.
2. Run Test programme ADC01 in Image BOARDTST.
3. Verify that counting pattern appears on {U12, U22, U32, U42-Pins 19, 18, 17, 16, 15, 14, 13, 12}.
4. Power Down and prepare for next test if operations OK.

1. Install U1 and apply +5V and +25V to Power board.
2. Run Test programme ADC02 in Image BOARDTST.
3. Verify that analogue Ramp pattern appears on U1-Pins 1, 7, 8, 14, 13, 12. and on Connectors J1, J2, J3, and J4.
4. Power Down and prepare for next test if operations OK.
5. All Test complete, Board now ready for service.
Construction and Test Procedures for the General Purpose Device Programmer.

Interface Connection Board (Specific).

If any parts of test procedure fails rectify problem then restart full test sequence.

1. Connect power to board +5V and +30 Verify power appears on correct IC Pins.
2. Verify voltage across D30 is 5.6 Volts. Using a 470Ω resistor use it to link U19,20 and U19,19 RED LED should switch on also U19,20 and U19,18 and GREEN LED should switch on.
3. Power Down and prepare for next test if operations OK.

1. Connect IF Board to Processor Board using (Normal leads 3 * 10/10 PORT_A=ADDR, PORT_B=PUT_DATA and PORT_C = GET_DATA)
2. Run Test programme IF01 in Image BOARDTST.(Only run this test if All ICs are removed)
3. Verify that counting patten appears on U10 & U11-Pins 2,3,4,5,6,7,8,9 and pulses appears on U19 Pin 15 also JRTC Pin 2 and on Pin 2 on all JP®M Connectors ( ⊗ is 1 to 8).

1. Install U10, U11 and apply +5V to Power board via JPOWER.
2. Run Test programme IF02 in Image BOARDTST.
3. Verify that count patten appears on {U10 & U11-Pins 18,17,16,15,14,13,12,11 } {U13 & U14-Pins 1,2,3 } {U12-Pins 2,3,13,14 } {U16, U17, U18 & U19-Pins 2,3,4,5,6,7,8,9 } {U15-Pins 2,3,5,6,8,9 } {U30-Pins 4,5,6,7,8,,9,10,11 }.
4. Power Down and prepare for next test if operations OK.

1. Install U12, U13, U14, U15, U17, U18 and apply +5V to Power board via JPOWER.
2. Run Test programme IF02 in Image BOARDTST.
3. Verify that count patten appears on {U17 & U18-Pins 18,17,16,15,14,13,12,11 } and pulses appears on {U15-Pins 18,17,16,15,14,12,11} {U13 & U14-Pins 7,9,10,11,12,13,14,15} {U12-Pins 4,5,7,10,11,12 } {U30-Pin 12 }.
4. Power Down and prepare for next test if operations OK.

1. Install U16, U19 and apply +5V to Power board via JPOWER.
2. Run Test programme IF03 in Image BOARDTST.
3. Verify that count patten appears on {U16 & U19-Pins 19,18,17,16,15,14,13,12 }.
4. Power Down and prepare for next test if operations OK.

1. Install U30, U31, Set RV30 to Mid Travel and apply +5V & +30V to Power board via JPOWER.
2. Check Voltage across D30 is 5.6Volts.
3. Run Test programme IF04 in Image BOARDTST.
4. Verify that a Ramp waveform appears on Test point JVREF-TP. Adjust RV30 till Maximum height ramp equal 25.5 Volts and Minimum Voltage is typically less that 0.2 Volts. Change R32 for more appropriate value if adjustment is at end of travel of RV30.
5. Run Test programme IF05 in Image BOARDTST if using DVM to calibrate.
6. Power Down and prepare for next test if operations OK.
7. All Test complete, Board now ready for service.
**Construction and Test Procedures for the General Purpose Device Programmer.**

**Important Pre Installation Instructions.**

If external clock module is **NOT** going to be installed prior to use then wire wrap JCLOCK pins 1, 2 and 3 together. Failure to complete this activity **WILL** make programming algorithms invalid.

**Document revision status.**

- Initial Preparation. 12th/Feb/2002
- ZIF03 Test programme update. 15th/Feb/2002
- Include actual used Processor Test Procedure. 17th/Feb/2002
- Include Additional Tests for OCTAL ZIF Board. 18th/Feb/2002
- Include Initial ADC Board Tests. 18th/Feb/2002
- Corrections to ADC Test procedure 1st/Mar/2002
- Include Initial Interface Board Tests. 9th/Apr/2002
- Include Table of Test Programme Start Address 10th/Apr/2002
- Procedure Corrections for ADC boards testing. 11th/Apr/2002
- Updates to Processor Serial Test programmes. 23th/Apr/2002
- Updated to Interface Board Checks 25th/Apr/2002
- . ..th/.../2002
- . ..th/.../2002
Programme Title is BOARDTST.ASM GENERAL BOARD TEST CODE PROGRAMMES

.Title BOARDTST.ASM General Board Test Code Programmes

General Note

These programmes are started via ZMON which is linked to this application.

To start a specific application use a Monitor GOTO - ORG address of section you wish to process.

...ASSEMBLER Z80+Extras

**********************************************************
**  Printer is FILE.TMP  File Being Processed is BOARDTST.LST                        On 06-May-2002   At 16:32:37       **
**********************************************************

Connect ZIF Board to processor board using a 10/2*5 cable to the Primary parallel ports B and C (Port B = LS 5 Bits Port C = MS bits) with a 10/6 cable to the Primary parallel port C1.

BIT allocation

Port B  Name  Port Bit  ZIF_BRD  Port B
26    Ground    Pin 1
27    SL1  Bit 0  Pin 1  Pin 2
28    SL2  Bit 1  Pin 2  Pin 3
29    SL4  Bit 2  Pin 3  Pin 4
30    GND  Bit 0  Pin 4  Pin 5
31    TBA  Bit 5  Pin 5  Pin 6
32    TBA  Bit 6  Pin 6  Pin 7
33    TBA  Bit 7  Pin 7  Pin 8
34    VCC  Pin 8  Pin 9
35    VCC  Pin 9  Pin 10
36    Port C  Name  Port Bit  ZIF_BRD  Port C
37    Ground  Pin 1
38    MC2  Bit 0  Pin 2  Pin 3
39    CL1  Bit 1  Pin 3  Pin 4
40    CL2  Bit 2  Pin 4  Pin 5
41    VV  Bit 3  Pin 5  Pin 6
42    VV  Bit 4  Pin 6  Pin 7
43    TBA  Bit 5  Pin 7  Pin 8
44    TBA  Bit 6  Pin 8  Pin 9
45    TBA  Bit 7  Pin 9  Pin 10
46    VCC  Pin 10
47    VCC  Pin 10
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106

Connect IF Main Board to processor board using a 3*10/10 cable to the Primary parallel Port A and JAD0R, Ports B and JPUT-DATA and Port C and JOUT-DATA.

BIT allocation

Port A  Name  Port Bit  IF_BRD  Port A
1    Ground  Pin 1  Pin 1
2    Address0  Bit 0  Pin 2  Pin 2
3    Address1  Bit 1  Pin 3  Pin 3
4    Address2  Bit 2  Pin 4  Pin 4
5    Address3  Bit 3  Pin 5  Pin 5
6    Address4  Bit 4  Pin 6  Pin 6
7    Address5  Bit 5  Pin 7  Pin 7
8    Address6  Bit 6  Pin 8  Pin 8
9    Address7  Bit 7  Pin 9  Pin 9

Interface Main BOARD Application Settings and Information.

**********************************************************
**  Printer is FILE.TMP  File Being Processed is JAD0R.LST                        On 06-May-2002   At 16:32:37       **
**********************************************************

Connect IF Main Board to processor board using a 3*10/10 cable to the Primary parallel Port A and JAD0R, Ports B and JPUT-DATA and Port C and JOUT-DATA.

BIT allocation

Port A  Name  Port Bit  IF_BRD  Port A
1    Ground  Pin 1  Pin 1
2    Address0  Bit 0  Pin 2  Pin 2
3    Address1  Bit 1  Pin 3  Pin 3
4    Address2  Bit 2  Pin 4  Pin 4
5    Address3  Bit 3  Pin 5  Pin 5
6    Address4  Bit 4  Pin 6  Pin 6
7    Address5  Bit 5  Pin 7  Pin 7
8    Address6  Bit 6  Pin 8  Pin 8
9    Address7  Bit 7  Pin 9  Pin 9

**********************************************************
**  Printer is FILE.TMP  File Being Processed is JPUT-DATA.LST                    On 06-May-2002   At 16:32:37       **
**********************************************************
107 ; 105 ; VCC N/C Pin 10
106 ; 106 ;
109 ; 107 ; Port B
110 ; 108 ; Port B Name Port Bit IF_BRD Port B
111 ; 109 ; Ground Pin 1 Pin 1
112 ; 110 ; Data0 Bit 0 Pin 2 Pin 2
113 ; 111 ; Data1 Bit 1 Pin 3 Pin 3
114 ; 112 ; Data2 Bit 2 Pin 4 Pin 4
115 ; 113 ; Data3 Bit 3 Pin 5 Pin 5
116 ; 114 ; Data4 Bit 4 Pin 6 Pin 6
117 ; 115 ; Data5 Bit 5 Pin 7 Pin 7
118 ; 116 ; Data6 Bit 6 Pin 8 Pin 8
119 ; 117 ; Data7 Bit 7 Pin 9 Pin 9
120 ; 118 ; VCC N/C Pin 10
121 ; 119 ;
122 ; 120 ; Port C
123 ; 121 ; Port B Name Port Bit IF_BRD Port C
124 ; 122 ; Ground Pin 1 Pin 1
125 ; 123 ; VM Bit 0 Pin 2 Pin 2
126 ; 124 ; SCI Bit 1 Pin 3 Pin 3
127 ; 125 ; TBA Bit 2 Pin 4 Pin 4
128 ; 126 ; TBA Bit 3 Pin 5 Pin 5
129 ; 127 ; Test Line Bit 4 Port A Pin 6 Pin 6
130 ; 128 ; TBA Bit 5 Pin 7 Pin 7
131 ; 129 ; TBA Bit 6 Pin 8 Pin 8
132 ; 130 ; TBA Bit 7 Pin 9 Pin 9
133 ; 131 ; VCC N/C Pin 10
134 ; 132 ;
135 ; 133 ;
136 ; 134 ;
137 ; 135 ; PORT DEFINITIONS SECTION
138 ; 136 ;
139 ; 137 ;
140 ; 138 ;
141 ; 139 ; PORTA = 40H ; A few variant PORT assignment Methods
142 ; 140 ; PORTB = 41H
143 ; 141 ; PORTC = PORTB + 1
144 ; 142 ; CONTROL = PORTA + 3
145 ; 143 ;
146 ; 144 ; ;--------------------------------------------------------------------------
147 ; 145 ; $X = 00000000H ; Port Definition Word
148 ; 146 ; ;--------------------------------------------------------------------------
149 ; 147 ; $X = $X + 00000000% ; Port B is Output
150 ; 148 ; $X = $X + 00000010% ; Port B is Input
151 ; 149 ; $X = $X + 01000000% ; Port A is Mode 0
152 ; 150 ; $X = $X + 00100000% ; Port A is Mode 1
153 ; 151 ; $X = $X + 00010000% ; Port C is Mode 0
154 ; 152 ; $X = $X + 00000000% ; Port C is Mode 1
155 ; 153 ; $X = $X + 10000000% ; Control Word is an I/O Mode Definition
156 ; 154 ; $X = $X + 10000000% ; Control Word is a Bit Set Definition
157 ; 155 ; $X = 00000000% ; Preset Definition Word
158 ; 156 ; ;--------------------------------------------------------------------------
159 ; 157 ; CONTRL = PORTA + 3
160 ; 158 ; PORTB = 41H
161 ; 159 ; ;--------------------------------------------------------------------------
162 ; 160 ; ; ;--------------------------------------------------------------------------
163 ; 161 ; ; ;--------------------------------------------------------------------------
164 ; 162 ; ; ;--------------------------------------------------------------------------
165 ; 163 ; ; ;--------------------------------------------------------------------------
166 ; 164 ; ; ;--------------------------------------------------------------------------
167 ; 165 ; ; ;--------------------------------------------------------------------------
168 ; 166 ; ; ;--------------------------------------------------------------------------
169 ; 167 ; ; ;--------------------------------------------------------------------------
170 ; 168 ; ; ;--------------------------------------------------------------------------
171 ; 169 ; ; ;--------------------------------------------------------------------------
172 ; 170 ; ; ;--------------------------------------------------------------------------
173 ; 171 ; ; ;--------------------------------------------------------------------------
174 ; 172 ; ; ;--------------------------------------------------------------------------
175 ; 173 ; ; ;--------------------------------------------------------------------------
176 ; 174 ; ; ;--------------------------------------------------------------------------
177 ; 175 ; ; ;--------------------------------------------------------------------------
178 ; 176 ; ; ;--------------------------------------------------------------------------
179 ; 177 ; ; ;--------------------------------------------------------------------------
180 ; 178 ; ; ;--------------------------------------------------------------------------
181 ; 179 ; ; ;--------------------------------------------------------------------------
182 ; 180 ; ZIF01:
183 ; 181 ; 1000 3E 80 182 ; 1000 3E 43 183 ; 1002 D3 43 184 ; 1004 3E 00 185 ; 1006 D3 40 186 ; 1008 D3 41 187 ; 100A D3 42 188 ; 100C 01 0000 189 ; 100F 78 190 ; 1010 D1 40 191 ; 1012 79 192 ; 1013 D3 42 193 ; 1016 3E 1F 194 ; 1018 A0 195 ; 101A 47 196 ; 101B 04 197 ; 101C C2 0F10 198 ; 101D DC 199 ; 101E 3E 1F 200 ; 1020 A1 201 ; 1021 4F 202 ; 1022 C3 0F10 203 ; 1023 FCF 0 204 ; 1024 0 205 ; 1025 0 206 ; 1026 0 207 ; 1027 0 208 ; 1028 0 209 ; 1029 0 210 ; 102A 0 211 ; 102B 0 212 ; .PAGE
177 ; 178 ; .ORG 1000
187 ; 188 ; CONTRL = PORTA + 3
188 ; 189 ; PORTB = 41H
189 ; 190 ; ; REPEAT FOREVER
191 ; 192 ; 100S 78 193 ; 1010 D1 40 194 ; 1012 79 195 ; 1013 D3 42 196 ; 1015 04 197 ; 1016 3E 1F 198 ; 1018 A0 199 ; 101A 47 200 ; 101B 04 201 ; 101C C2 0F10 202 ; 101D DC 203 ; 101E 3E 1F 204 ; 1020 A1 205 ; 1021 4F 206 ; 1022 C3 0F10 207 ; 1023 FCF 0 208 ; 1024 0 209 ; 1025 0 210 ; 1026 0 211 ; 1027 0 212 ; .PAGE
187 ; 188 ; out (PortA),A ; Write Binary Patten to 8255
188 ; 189 ; out (PortC),A ; Write Binary Patten to 8255
189 ; 190 ; out (PortB),A ; Write Binary Patten to 8255
190 ; 191 ; LD A,#0 ; HOLD CHANGE BITS
191 ; 192 ; out (PortB),A ; Write Binary Patten to 8255
192 ; 193 ; ; ; STOP
193 ; 194 ; LD A,C ; HOLD CHANGE BITS
194 ; 195 ; out (PortC),A ; Write Binary Patten to 8255
195 ; 196 ; ; ; STOP
196 ; 197 ; INC B ; UPDATE CHANGE BITS
197 ; 198 ; LD A,#1F ; SET UP LIMIT MASK
198 ; 199 ; AND B ; LIMIT RANGE
200 ; 201 ; LD R,A ; KEEP UPDATED VALUE
201 ; 202 ; JP NL,100S ; IF SLOW UPDATE REQUIRED
202 ; 203 ; THEN
203 ; 204 ; INC C ; UPDATE CHANGE BITS
204 ; 205 ; LD A,#1F ; SET LIMIT LIMIT MASK
205 ; 206 ; AND C ; LIMIT RANGE
206 ; 207 ; LD C,A ; KEEP UPDATED VALUE
207 ; 208 ; ; ENDIF
208 ; 209 ; JP 1005 ; END REPEAT
209 ; 210 ; ; ; STOP
210 ; 211 ; ; ; STOP
211 ; 212 ; ; ; STOP
212 ; 213 ; .SBTTL . Application ZIF02 SECTION
213 ; 214 ; ; ; STOP
214 ; 215 ; ; ; STOP
215 ; 216 ; ; ; STOP
;Write Binary Pattern to 8255 (BOTTOM 5 BITS)
.out (PortA),A

;Write Binary Pattern to 8255
.out (PortB),A

;Write Binary Pattern to 8255
.out (PortC),A

;START BIT TO PROCESS
.LD C,#0

;REPEAT FOR EACH BIT
.LD A,#00000000%

;MC1 IS INACTIVE (BIT 0)
.LD A,#00011110%

;MC2 IS INACTIVE (BIT 1)
.LD A,#00100000%

;MC3 IS INACTIVE (BIT 2)
.LD A,#00101000%

;MC4 IS INACTIVE (BIT 3)
.LD A,#01001000%

;MC5 IS INACTIVE (BIT 4)
.LD A,#01011000%

;MC6 IS INACTIVE (BIT 5)
.LD A,#10000000%

;MC7 IS INACTIVE (BIT 6)
.LD A,#10001000%

;MC8 IS INACTIVE (BIT 7)
.LD A,#10011000%

;DATA TRANSFER
.LD A,#0

;PIO Control Port Settings Assignment
.LD A,#$X

;MSB VOLT = 4
.LD A,#4

;MC1 IS INACTIVE (BIT 4)
.LD A,#10000000%

;MC1 IS INACTIVE (BIT 1)
.LD A,#00100000%

;MC2 IS INACTIVE (BIT 2)
.LD A,#01000000%

;MC3 IS INACTIVE (BIT 3)
.LD A,#10000000%

;MC4 IS INACTIVE (BIT 4)
.LD A,#10001000%

;MC5 IS INACTIVE (BIT 5)
.LD A,#10010000%

;MC6 IS INACTIVE (BIT 6)
.LD A,#10011000%

;MC7 IS INACTIVE (BIT 7)
.LD A,#11000000%

;MC8 IS INACTIVE (BIT 8)
.LD A,#11001000%

;DATA TRANSFER
.LD A,#0

;PIO Control Port Settings Assignment
.LD A,#$X

;MSB VOLT = 4
.LD A,#4

;MC1 IS INACTIVE (BIT 4)
.LD A,#10000000%

;MC1 IS INACTIVE (BIT 1)
.LD A,#00100000%

;MC2 IS INACTIVE (BIT 2)
.LD A,#01000000%

;MC3 IS INACTIVE (BIT 3)
.LD A,#10000000%

;MC4 IS INACTIVE (BIT 4)
.LD A,#10001000%

;MC5 IS INACTIVE (BIT 5)
.LD A,#10010000%

;MC6 IS INACTIVE (BIT 6)
.LD A,#10011000%

;MC7 IS INACTIVE (BIT 7)
.LD A,#11000000%

;MC8 IS INACTIVE (BIT 8)
.LD A,#11001000%

;DATA TRANSFER
.LD A,#0

;PIO Control Port Settings Assignment
.LD A,#$X

;MSB VOLT = 4
.LD A,#4

;MC1 IS INACTIVE (BIT 4)
.LD A,#10000000%

;MC1 IS INACTIVE (BIT 1)
.LD A,#00100000%

;MC2 IS INACTIVE (BIT 2)
.LD A,#01000000%

;MC3 IS INACTIVE (BIT 3)
.LD A,#10000000%

;MC4 IS INACTIVE (BIT 4)
.LD A,#10001000%

;MC5 IS INACTIVE (BIT 5)
.LD A,#10010000%

;MC6 IS INACTIVE (BIT 6)
.LD A,#10011000%

;MC7 IS INACTIVE (BIT 7)
.LD A,#11000000%

;MC8 IS INACTIVE (BIT 8)
.LD A,#11001000%

;DATA TRANSFER
.LD A,#0

;PIO Control Port Settings Assignment
.LD A,#$X

;MSB VOLT = 4
.LD A,#4

;MC1 IS INACTIVE (BIT 4)
.LD A,#10000000%

;MC1 IS INACTIVE (BIT 1)
.LD A,#00100000%

;MC2 IS INACTIVE (BIT 2)
.LD A,#01000000%

;MC3 IS INACTIVE (BIT 3)
.LD A,#10000000%

;MC4 IS INACTIVE (BIT 4)
.LD A,#10001000%

;MC5 IS INACTIVE (BIT 5)
.LD A,#10010000%

;MC6 IS INACTIVE (BIT 6)
.LD A,#10011000%

;MC7 IS INACTIVE (BIT 7)
.LD A,#11000000%

;MC8 IS INACTIVE (BIT 8)
.LD A,#11001000%

;DATA TRANSFER
.LD A,#0

;PIO Control Port Settings Assignment
.LD A,#$X

;MSB VOLT = 4
.LD A,#4

;MC1 IS INACTIVE (BIT 4)
.LD A,#10000000%

;MC1 IS INACTIVE (BIT 1)
.LD A,#00100000%

;MC2 IS INACTIVE (BIT 2)
.LD A,#01000000%

;MC3 IS INACTIVE (BIT 3)
.LD A,#10000000%

;MC4 IS INACTIVE (BIT 4)
.LD A,#10001000%

;MC5 IS INACTIVE (BIT 5)
.LD A,#10010000%

;MC6 IS INACTIVE (BIT 6)
.LD A,#10011000%

;MC7 IS INACTIVE (BIT 7)
.LD A,#11000000%

;MC8 IS INACTIVE (BIT 8)
.LD A,#11001000%

;DATA TRANSFER
.LD A,#0

`LD DE, #OFFF`; ;DELAY VALUE

`LD A, #D`; ;LOOP TILL COUNT EXPIRED

`LD A, #E`; ;OR E

`LD A, #H`; ;JP NZ, $3500

`LD A, #00000000`; ;All Functions Inactive

`LD A, #C6`; ;CHANNEL CONTROL MASK

`LD A, #7`; ;Extraction Mask for channel number

`LD A, #E`; ;Get the Channel to Process

`LD A, #B`; ;If Immediate Data transfer reqd

`LD A, #585`; ;THEN

`LD A, #585`; ;END
Select the feature 435  

Always ensure the board is selected 435

Write Binary Patten to 8255 435

Hold the Data Patten 440

Write Binary Patten to 8255 441

Remember previous channel settings 443

Process all Channels 444

Remember previous counter settings 446

Process all Data Patterns 447

Restart everything again 449
657  254F  C2 4825  655  JP NE,25200  ;Wait till delay exhausted
659                                                                                      
662  2552  3E 7F  660  LD A,#127.  ;Select Half Max Volt Reference
664  2554  3E C0  662  LD A,#000H  ;Select OV Reference
666  255D  D3 41  663  out (PortB),A  ;Write Binary Patten to 8255
666  255E  3E 80  664  LD A,#080H  ;Select Reference 8
668  255A  D3 40  666  out (PortA),A  ;Write Binary Patten to 8255
669  255C  3E FF  667  LD A,#0FFH  ;DeSelect U19
670  255E  D3 40  668  out (PortA),A  ;Write Binary Patten to 8255
671                                                                                      
672  2560  3E 01  671  LD A,#001H  ;Select RED LED ON , GREEN LED OFF
674  2562  D3 41  672  out (PortB),A  ;Write Binary Patten to 8255
675                                                                                      
676  2564  3E 80  674  LD A,#0E0H  ;Select U19
676  2566  D3 40  675  out (PortA),A  ;Write Binary Patten to 8255
676  2568  3E FF  677  LD A,#0CFH  ;DeSelect U19
678  256A  D3 40  678  out (PortA),A  ;Write Binary Patten to 8255
679                                                                                      
682  256C  D1 0000  680  LD BC,#0  ;Long Delay period
683  256F  DB  681  2530H; DEC BC  ;Reduce Delay Counter
684  2570  00  682  NOP  ;Extra Delay
685  2571  00  683  NOP  ;Extra Delay
686  2572  00  684  NOP  ;Extra Delay
687  2573  00  685  NOP  ;Extra Delay
688  2574  00  686  LD A,B  ;Hold High Byte
689  2575  01  687  OR C  ;Merge Low Byte
689  2576  C2 6P25  688  JP NE,25300  ;Wait till delay exhausted
691  2577  00  689                                                                                      
692                                                                                      
693  2579  3E FF  693  LD A,#0FFH  ;Select Max Volt Reference
695                                                                                      
697  257B  D3 41  694  out (PortB),A  ;Write Binary Patten to 8255
697  257C  01  695                                                                                      
700  2581  3E FF  698  LD A,#003H  ;Select RED LED ON , GREEN LED ON
700  2583  D3 40  699  out (PortA),A  ;Write Binary Patten to 8255
701                                                                                      
702  2585  3E 03  701  LD A,#003H  ;Select RED LED ON , GREEN LED ON
704  2587  D3 41  703  out (PortB),A  ;Write Binary Patten to 8255
706  2589  3E 00  705  LD A,#0E0H  ;Select U19
706  258B  D3 40  706  out (PortA),A  ;Write Binary Patten to 8255
709                                                                                      
710  258D  3E FF  709  LD A,#005H  ;Select RED LED ON, GREEN LED OFF
711  258F  D3 40  709  out (PortA),A  ;Write Binary Patten to 8255
712                                                                                      
713  2591  D1 0000  711  LD BC,#0  ;Long Delay period
715  2594  08  712  2540H; DEC BC  ;Reduce Delay Counter
715  2595  00  713  NOP  ;Extra Delay
716  2596  00  714  NOP  ;Extra Delay
717  2596  00  715  NOP  ;Extra Delay
718  2598  00  716  NOP  ;Extra Delay
718  2599  78  717  LD A,B  ;Hold High Byte
720  259B  B1  718  OR C  ;Merge Low Byte
721  259B  C2 9425  719  JP NE,25400  ;Wait till delay exhausted
722                                                                                      
723  259E  C3 0825  721  JP 25000  ;START AGAIN
724                                                                                      
725  25A2  00  722                                                                                      
726  25A2  A=2600  723                                                                                      
727  09  2600  724                                                                                      
730                                                                                      
731  2600  3E 88  730  LD A,#5Y  ;PIO Control Port Settings Assignment
733  2602  D3 40  731  out (PortA),A  ;Configure 2555
733  2604  3E FF  732  LD A,#0FFH  ;DeSelect Everything
735  2606  D3 40  733  out (PortA),A  ;Write Binary Patten to 8255
736                                                                                      
737  2608  3E 00  736  LD A,#000H  ;Select OV Reference
739  260A  CD 3826  737  CALL 2650  ;Set voltage
740  738                                                                                      
741  260D  3E 01  739  LD A,#001H  ;Select Reference 1
740  260F  CD 3826  740  CALL 2650  ;Set voltage
741                                                                                      
744  2612  3E 02  743  LD A,#002H  ;Select Reference 2
745  2614  CD 3826  744  CALL 2650  ;Set voltage
746                                                                                      
747  2617  3E 04  745  LD A,#004H  ;Select Reference 3
748  2619  CD 3826  746  CALL 2650  ;Set voltage
749                                                                                      
750  261C  3E 08  747  LD A,#008H  ;Select Reference 4
752  261E  CD 3826  748  CALL 2650  ;Set voltage
753                                                                                      
755  2621  3E 10  752  LD A,#010H  ;Select Reference 5
756  2623  CD 3826  753  CALL 2650  ;Set voltage
755  2625  3E 14  753  LD A,#014H  ;Select Reference 6
757  2627  CD 3826  754  CALL 2650  ;Set voltage
756  2628  3E 20  755  LD A,#020H  ;Select Reference 7
758  262B  CD 3826  756  CALL 2650  ;Set voltage
759                                                                                      
761  2630  3E 24  760  LD A,#000H  ;Select Reference 8
762  2632  CD 3826  761  CALL 2650  ;Set voltage
764                                                                                      
765  2635  C3 0826  763  GOTO 2600  ;Start again
766                                                                                      

767  2638  D3 41  768  26505;  769  out (PortB),A  770  ; Write Binary Patten to 8255
771  263A  3E C0  772  LD A,#0C0H  773  ;Select U30
774  263C  D3 40  775  out (PortA),A  776  ; Write Binary Patten to 8255
777  263E  D3 40  778  LD A,#0FFH  779  ;DeSelect U30
780  2640  3E FF  781  out (PortA),A  782  ; Write Binary Patten to 8255
783  2642  D3 40  784  LD BC,#0  785  ;Long Delay period
786  2644  01 0000  787  DEC BC  788  ;Reduce Delay Counter
789  2647  0B  789  2660$:  DEC BC  790  ;Extra Delay
791  2648  00  792  NOP  793  ;Extra Delay
794  2649  00  795  NOP  796  ;Extra Delay
797  264A  00  798  NOP  799  ;Extra Delay
799  264B  00  800  NOP  801  ;Hold High Byte
802  264C  78  803  LD A,B  804  ;Merge Low Byte
805  264D  C2 4726  806  JP NZ,2660$  807  ;Wait till delay exhausted
808  2651  C9  809  RET  810  ;Back to caller
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
.Title Z80TST.ASM Binary counting using 8255 Device with LEDs on Port B

; Definitions Section

PORTA = 80H ; A few variant PORT assignment methods
PORTB = 81H
PORTC = PORTB + 1
CONTROL = PORTA + 3

16 ld SP, #51Addr

; DESCRIPTION
; THIS ROUTINE WILL CREATE A DELAY OF 1 TO 64K MILLI-SECONDS
; THE TIME OF THE CALL TO THIS ROUTINE AND ITS RETURN IS TAKEN INTO ACCOUNT AS THE TIMING PERIOD.
; ENTRY CONDITIONS
; BC = NUMBER OF MILLISECONDS TO DELAY
; EXIT CONDITIONS
; NONE

; AUTHOR: R. J. SPRIGGS

52 LD A, #89H ; See Above for 8255 Mode BIT settings
53 LD A, #257. ; Preferred/Alternative Assignment Method
54 ld A, #89H ; Preferred/Alternative Assignment Method

57 .ASSEMBLER Z80+EXTRAS
58 .ORG 0
59 .ASSEMBLER Z80TST.ASM Binary counting using 8255 Device with LEDs on Port B
59 *************************************************** *************************************************** ******************
59 * Printer is FILE.TMP File Being Processed is Z80 TST.ASM                          On 24-Apr-2002   At 15:55:55       *
59 *************************************************** *************************************************** **************

; .PAGE
61 .SBTTL HALT PROGRAM FOR A NUMBER OF MILLI-SECONDS
62 .GAP 20h
63
64 SMSEC:
65
66 ;
67 ; DESCRIPTION
68 ; THIS ROUTINE WILL CREATE A DELAY OF 1 TO 64K MILLI-SECONDS
69 ; THE TIME OF THE CALL TO THIS ROUTINE AND ITS RETURN IS TAKEN INTO ACCOUNT AS THE TIMING PERIOD.
70 ; ENTRY CONDITIONS
71 ; BC = NUMBER OF MILLISECONDS TO DELAY
72 ; EXIT CONDITIONS
73 ; NONE
74 ; AUTHOR: R. J. SPRIGGS
75 ;
76 CALL SMSEC ; 17T ; TIME TO CALL THIS ROUTINE
77 PUSH AF ; 11T ; PROTECT ALL REGISTERS
78 PUSH BC ; 11T
79 PUSH DE ; 11T
80
81 LD DE, #78. ; 10T ; TIMMING PERIOD COUNTER IF ONLY ONE PASS
82 ; 80T ; TOTAL FOR SECTOR 'A'
83 10S: NOP ; 4T ; TIMMING PAD
84 ; 4T ; TOTAL FOR SECTOR 'B'
85 91
86 20S: DEC DE ; 6T ; REDUCE PERIOD COUNTER
87 LD A, D ; 4T ; HOLD HIGH BYTE
88 OR E ; 4T ; CHECKSUM WITH LOW BYTE
89 JP NZ, 20S ; 10T ; LOOP TILL TIMING COUNTER EXHAUSTED
90 ; 24T ; TOTAL FOR SECTOR 'C'
91 97
92 DEC BC ; 6T ; REDUCE REPEAT COUNTER
93 LD A, B ; 4T ; HOLD HIGH BYTE
94 OR C ; 4T ; CHECKSUM WITH LOW BYTE
95 JP Z, 20S ; 10T ; EXIT IF REPEAT COUNTER EXHAUSTED
96 ; 24T ; TOTAL FOR SECTOR 'D'
97
98 104 LD DE, #81. ; 10T ; INTERNAL RESET FOR PERIOD COUNTER
99 NOP ; 4T ; TIMMING PAD
100 NOP ; 4T ; TIMMING PAD
101
JP 203 ;10T ;ENTER DELAY LOOP AGAIN
30$: POP DE ;10T ;TOTAL FOR SECTOR 'E'
POP BC ;10T
POP AF ;10T
RET ;10T ;BACK TO CALLER
40T ;TOTAL FOR SECTOR 'F'

; TIMING CALCULATIONS

; WHERE :-
; 2T STATES = 1 MICRO SECOND

A = 60T STATES
B = 4T STATES
C = 24T STATES
D = 24T STATES
E = 28T STATES
F = 40T STATES

; ASSUME 'BC' = 1
; TIME = A + B + 78C + D + F
; 60 + 4 + 1872 + 24 + 40 = 2000

; ASSUME 'BC' = N
; TIME = A + B + 78C + D + F + (N-1)(E + B + 81C + D)
; 60 + 4 + 1872 + 24 + 40 + (N-1)(4 + 28 + 1944 + 24) = (N-1)(2000)

; ORG 1800
.reserve 30 ; Reserve space for Stack
; Note Stack operates top of memory down
.end ; of programme
MOV R6,#1           ; Set Direction Left
CJNE R7,#01,wait    ; If end Reached
RR A                ; Move Pattern Right
RIGHT:
SJMP WAIT
MOV R6,#0           ; Set Direction Right
CJNE R7,#80,wait    ; If end Reached
MOV R7,A            ; Hold Display Pattern
CJNE R6,#1,RIGHT    
LOOP:
MOV A,R7           ; Master Display Pattern
MOVX @DPTR,A        ; Display Pattern
CJNE R6,#1,RIGHT    
LOOP:
MOV A,R7           ; Light Display Pattern
MOVX @DPTR,A        ; Display Pattern
CJNE R6,#1,RIGHT    
LOOP:
RL A                ; Move Pattern Left
MOV A,R7           ; Hold Display Pattern
CJNE R7,#0,wait    ; If end Reached
MOV A,R0           ; Set Direction Right
SJMP WAIT
RIGHT:
RR A                ; Move Pattern Right
MOV A,R7           ; Hold Display Pattern
CJNE R7,#0,wait    ; If end Reached
MOV A,R0           ; Set Direction Left
WAIT:   ; ACALL DELAY ; Delay Removed to give quick Simulator Demo
109      SJMP LOOP ; Loop Forever
111
113 DELAY: ; Sleep for 3 * 1/10 Secs Period
114        MOV R2,#3
115 DELAY1: ; Sleep for 1/10 Secs Delay
116        MOV R1,#200.
117 DELAY3: ; Sleep for 1/2 msec Delay
118        MOV R0,#249.
119        NOP
120 DELAY5:
121        DJNZ R0,DELAY5
122        DJNZ R1,DELAY3
123        DJNZ R2,DELAY1
124        RET
125
127 . END ; Source Complete
128
129 ; Nothing is processed after the .END Directive
130
132 ; Assembler variations and Notes
134 ; 1. NO EQU directive So use SYMBOL = <Expression> structure
136 ; 2. ORG is prefixed with a dot ie. .ORG <Address>
138 ; 3. Default RADIX is 16 (Can be altered using .RADIX directive)
140 ; 4. Note SYMBOL and LABEL names MAX 6 Characters
142 ; 5. Note #?? implies a Constant
144 ; 6. Note $?? implies a Label or Symbol
146 ; 7. Currently no Generic JMP order hence must use Specifics
148 ; ie. SJMP , LJMP , AJMP
107 ; * GOTO   LABEL    ;GOTO   k               Where k = #11
108 ; * INCF  $file1,Dest.W ;INCF   f,d          Where f = #7
109 ; * INCFSZ $file1,Dest.f  ;INCFSZ f,d       Where f = #7
110 ; * IORLW $SPOS      ;IORLW  k               Where k = #8
111 ; * IORWF  $file2,Dest.f  ;IORWF f,d        Where f = #7
112 ; * INCF $file1,Dest.W   ;INCF   f,d          Where f = #7
113 ; * INCF $file1,Dest.f   ;INCF   f,d          Where f = #7
114 ; * IORLW $POS        ;IORLW  k               Where k = #8
115 ; * IORWF  $file2,Dest.f  ;IORWF f,d        Where f = #7
116 ; * MOVLW 55.         ;MOVLW  k               Where k = #8
117 ; * MOVF  $file2,Dest.f  ;MOVWF f,d        Where f = #7
118 ; * MOVLW 88.         ;MOVLW  k               Where k = #8
119 ; * MOVF  $file2,Dest.f  ;MOVWF f,d        Where f = #7
120 ; * RETFIE             ;RETFIE
121 ; * NOP                ;NOP
122 ; * RETFIE             ;RETFIE
123 ; * RETLW 55.         ;RETLM  k               Where k = #8
124 ; * RETLW 88.         ;RETLM  k               Where k = #8
125 ; * RETLW 88.         ;RETLM  k               Where k = #8
126 ; * SUBWF  $POS        ;SUBWF  f,d          Where f = #7
127 ; * SUBWF  $POS        ;SUBWF  f,d          Where f = #7
128 ; * SUBWF  $POS        ;SUBWF  f,d          Where f = #7
129 ; * SUBWF  $POS        ;SUBWF  f,d          Where f = #7
130 ; * RLF   $file2,Dest.w   ;RLF    f,d         Where f = #7
131 ; * RLF   $file2,Dest.w   ;RLF    f,d         Where f = #7
132 ; * XORLW $NEG         ;XORLW  k               Where k = #8
133 ; * XORLW $NEG         ;XORLW  k               Where k = #8
134 ; * XORLW $NEG         ;XORLW  k               Where k = #8
135 ; * XORLW $NEG         ;XORLW  k               Where k = #8
136 ; * XORLW $NEG         ;XORLW  k               Where k = #8
137 ; * XORLW $NEG         ;XORLW  k               Where k = #8
138 ; * XORLW $NEG         ;XORLW  k               Where k = #8
139 ; * XORLW $NEG         ;XORLW  k               Where k = #8
140 ; " .REGION REGISTERS
141 ; " .ORG    0000H      ;START ADDRESS OF FILE DATA
142 ; " .reserve 20        ;Note this area is not presettable
143 ; " .reserve 11        ;To preset check assembler ONLY
144 ; " .reserve 2
145 ; " .END
APPENDIX
Sub-Section

This section contains the following item:

Application Specific Language Source code for PIC16

Description:
This is the Application Specific Language Source code for the General Purpose Cross Assembler. This Source is specifically targeted for the PIC16C84 processor, however, should be suitable for the whole of the PIC16xxxx series processors.

<table>
<thead>
<tr>
<th>Title of Pages</th>
<th>Number of Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation Files.</td>
<td>3</td>
</tr>
<tr>
<td>The Cross Assembler Application Specific Language code for the PIC16C84. Filename PIC16.LDS</td>
<td>3</td>
</tr>
</tbody>
</table>
BEGIN

14    BEGIN
15        META IF_PARM 1=CLRW       : BEGIN : DO "01 03"    : EXIT_HEX : END
16        META IF_PARM 1=CLRWDT     : BEGIN : DO "00 64"    : EXIT_HEX : END
17        META IF_PARM 1=NOP        : BEGIN : DO "00 00"    : EXIT_HEX : END
18        META IF_PARM 1=RETFIE     : BEGIN : DO "00 09"    : EXIT_HEX : END
19        META IF_PARM 1=RETURN     : BEGIN : DO "00 08"    : EXIT_HEX : END
20        META IF_PARM 1=SLEEP      : BEGIN : DO "00 63"    : EXIT_HEX : END
21    IF_PARM 1=~~ID_MAIN
22          META BEGIN : EXIT_CODE "PIC16 03/09/01 V1.02" : END
23
24    EXIT_ERROR Unknown Opcode
25    RESTART LIBRARY_END
26    END

IF_COUNT 2

BEGIN

35    IF_PARM 1=ADCLM
36        META BEGIN : DO "03EH",2="{+255,-128}"    : EXIT : END
37    IF_PARM 1=ANDLW
38        META BEGIN : DO "039H",2="{+255,-128}"    : EXIT : END
39    IF_PARM 1=CALL
40        META BEGIN : DO "001H","080H+"2="{+255,+128}"  : EXIT : END
41    IF_PARM 1=CLRF
42        META BEGIN : DO "00AH",3="*128.+2="{+255,+0}" : EXIT : END
43    IF_PARM 1=DECFSZ
44        META BEGIN : DO "00BH",3="*128.+2="{+255,+0}" : EXIT : END
45    IF_PARM 1=INCF
46        META BEGIN : DO "00CF",3="*128.+2="{+255,+0}" : EXIT : END
47    IF_PARM 1=~~INS_WIDTH
48        BEGIN
49          IF_PARM 2=ISPACE
50            META BEGIN : EXIT_CODE 2 : END
51    END
52    EXIT_ERROR Unknown Opcode
53    RESTART LIBRARY_END
54    END

IF_COUNT 3

BEGIN

71    IF_PARM 2=ISPACE
72        META BEGIN : EXIT_CODE 2 : END
73        RESTART LIBRARY_END
74    END
75    ; EXIT_ERROR Unknown Opcode
76    RESTART LIBRARY_END
77    END
78    END
79    ;******************************************************************************
80
81    IF_COUNT 4
82    BEGIN
83
84    IF_PARM 1=ADWF
85        META BEGIN : DO "007H",3="*128.+2="{+255,+0}" : EXIT : END
86    IF_PARM 1=ANDWF
87        META BEGIN : DO "005H",3="*128.+2="{+255,+0}" : EXIT : END
88    IF_PARM 1=COMP
89        META BEGIN : DO "009H",3="*128.+2="{+255,+0}" : EXIT : END
90    IF_PARM 1=DECFSZ
91        META BEGIN : DO "00AH",3="*128.+2="{+255,+0}" : EXIT : END
92    IF_PARM 1=INCF
93        META BEGIN : DO "00CH",3="*128.+2="{+255,+0}" : EXIT : END
94    IF_PARM 1=DECFSZ
95        META BEGIN : DO "00BH",3="*128.+2="{+255,+0}" : EXIT : END
96    IF_PARM 1=INCF
97        META BEGIN : DO "009H",3="*128.+2="{+255,+0}" : EXIT : END
98    IF_PARM 1=DECFSZ
99        META BEGIN : DO "00CH",3="*128.+2="{+255,+0}" : EXIT : END
100   IF_PARM 1=INCF
101      META BEGIN : DO "00BH",3="*128.+2="{+255,+0}" : EXIT : END
102   IF_PARM 1=DECFSZ
103      META BEGIN : DO "009H",3="*128.+2="{+255,+0}" : EXIT : END
104   IF_PARM 1=INCF
105      META BEGIN : DO "00CH",3="*128.+2="{+255,+0}" : EXIT : END
106   IF_PARM 1=DECFSZ
107
META BEGIN : DO "00FH",3="*128.+ "2="#255\+0" : EXIT : END

IF_PARM 1=IORWF
META BEGIN : DO "004H",3="*128.+ "2="#255\+0" : EXIT : END

IF_PARM 1=MOVF
META BEGIN : DO "008H",3="*128.+ "2="#255\+0" : EXIT : END

IF_PARM 1=RLF
META BEGIN : DO "00DH",3="*128.+ "2="#255\+0" : EXIT : END

IF_PARM 1=RRF
META BEGIN : DO "00CH",3="*128.+ "2="#255\+0" : EXIT : END

IF_PARM 1=SUBWF
META BEGIN : DO "002H",3="*128.+ "2="#255\+0" : EXIT : END

IF_PARM 1=SWAPF
META BEGIN : DO "00EH",3="*128.+ "2="#255\+0" : EXIT : END

IF_PARM 1=XORWF
META BEGIN : DO "006H",3="*128.+ "2="#255\+0" : EXIT : END

BIT OPERATIONS

IF_PARM 1=BCF
BEGIN
META DO "( "3="/2+10H )+(19\+16)" ,3="401M\+128.+"2="#255\+0" : EXIT
END

IF_PARM 1=BSF
BEGIN
META DO "( "3="/2+14H )+(23\+20)" ,3="401M\+128.+"2="#255\+0" : EXIT
END

IF_PARM 1=BTFSC
BEGIN
META DO "( "3="/2+18H )+(27\+24)" ,3="401M\+128.+"2="#255\+0" : EXIT
END

IF_PARM 1=BTFSS
BEGIN
META DO "( "3="/2+1CH )+(31\+28)" ,3="401M\+128.+"2="#255\+0" : EXIT
END

EXIT_ERROR Unknown Opcode
END

RESTART LIBRARY_END
EXIT_ERROR Unknown Opcode
END
APPENDIX
Sub-Section

This section contains the following item:

Application Specific Language Source code for PIC16SIM

Description:

This is the Application Specific Language Source code for the General Purpose Simulator. This Source is specifically targeted for the PIC16C84 processor, however, it should be suitable for simulation of the basic code for some of the other processors in PIC16xxxx series range. Currently NO device specific implementation code has been included in this source, however, the access handles are available.

Title of Pages

<table>
<thead>
<tr>
<th>Documentation Files.</th>
<th>Number of Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Simulator Application Specific Language code for the PIC16C84. Filename PIC16SIM.LDS</td>
<td>26</td>
</tr>
</tbody>
</table>
1 ; Filename is Pic16SIM.LDS  Author R. J. Spriggs on 28/08/01 Updated 08/09/01
2 ; START DESIGN V1.01 RJS 01/09/01
3 ; Modification: Registers accessed via procedure V1.02 RJS 08/09/01
4 ; Modification: More Instructions Added V1.03 RJS 16/09/01
5 ; Modification: More Instructions Added V1.04 RJS 17/09/01
6 ; Modification: More Instructions Added V1.05 RJS 18/09/01
7 ; Modification: General Source Tidy V1.06 RJS 19/09/01
8 ; Modification V1.10 RJS ../../01
9 ; Pic16 Library for General Purpose Simulator
10 ;*************************************** *************************
11 ;             SINGLE PARAMETER ORDERS
12 ;*************************************** *************************
13 IF_PARM 1=~~ID_MAIN
14 BEGIN
15 EXIT_CODE Pic16SIM 19/09/01 V1.06 Seq 0067
16 END
17 IF_PARM 1=PROCESSOR
18 BEGIN
19 IF_FOUND 2,DEBUG
20 BEGIN
21 TRACE "DEBUG PARAMETER LOCATED"
22 END
23 ; Regular Initialisation operations
24 DO TS00IS01
25 ; Always the First Operation
26 DO GET_OP_CODE
27 ; \; H = High Op_Code  PC=PC+1 Q=PCH  P=PCL
28 ; \; M=(PCH) L=(PCL)
29 ;
30 ; Regular Initialisation operations
31 ; Register "I" Reset I=Inc/Dec Value = 1
32 ; Register "T" Reset Time State Transitions Counter
33 DO TS00IS01
34 ;
35 PERFORM GET_OP_CODE
36 ; \; H = High Op_Code  PC=PC+1 Q=PCH  P=PCL
37 ; \; M=(PCH) L=(PCL)
38 ;
39 ;
40 ;
41 ;
42 ;
43 ;
44 ;
45 ;
46 DO 0S30H4
47 IF_REG 0=00
48 BEGIN
49 IF_REG H=00
50 BEGIN
51 ;
52 ; This should be one of the following operations
53 ; \; MOVWF NOP CLRWDT RETFIE RETURN SLEEP
54 ;
55 ;
56 DO 0S9F4L
57 IF_REG 0=00
58 BEGIN
59 META IF_FOUND 2,DECOMP : BEGIN : DO 'NOP' : END
60 RESTART GENERAL_EXIT
61 END
62 DO 0S9F4L
63 IF_REG 0=80
64 BEGIN
65 DO 0S7FLB
66 ;\; META IF_FOUND 2,DECOMP : BEGIN : DO 'MOVWF 'BH : END
67 CACHE_SEL 3
68 CACHE_LOAD 02
69 CACHE_RA
70 PERFORM PUT_REGFILE
71 RESTART GENERAL_EXIT
72 END
73 IF_REG L=64
74 BEGIN
75 ;\; META_IF_FOUND 2,DECOMP : BEGIN : DO 'CLRWDT' : END
76 RESTART GENERAL_EXIT
77 END
78 IF_REG L=69
79 BEGIN
80 ;\; META_IF_FOUND 2,DECOMP : BEGIN : DO 'RETFIE' : END
81 PERFORM POP
82 ;\; MORE CODE STILL REQUIRED HERE
83 RESTART GENERAL_EXIT
84 END
85 IF_REG L=63
86 BEGIN
87 ;\; META_IF_FOUND 2,DECOMP : BEGIN : DO 'RETURN' : END
88 PERFORM POP
89 RESTART GENERAL_EXIT
90 END
91 IF_REG L=09
92 BEGIN
93 ;\; META_IF_FOUND 2,DECOMP : BEGIN : DO 'SLEEP' : END
94 ;\; PC = PC - 1
95 DO PSG-PP
96 ;\; META_IF_REG_LOWER P,00 : BEGIN : DO 0SFFP4PQG1-QP0S1FQ&P : END
97 RESTART GENERAL_EXIT
98 END
99 IF_REG L=08
100 BEGIN
101 ;\; META_IF_FOUND 2,DECOMP : BEGIN : DO 'RETURN' : END
102 PERFORM POP
103 RESTART GENERAL_EXIT
104 END
105 IF_REG L=00
106 BEGIN
107 ;\; META_IF_FOUND 2,DECOMP : BEGIN : DO 'RETIE' : END
108 PERFORM POP
109 ;\; MORE CODE STILL REQUIRED HERE
110 RESTART GENERAL_EXIT
111 END
112 IF_REG L=08
113 BEGIN
114 ;\; META_IF_FOUND 2,DECOMP : BEGIN : DO 'SLEEP' : END
115 ;\; PC = PC - 1
116 DO SG-PP
BEGIN
IF_REG H=01 BEGIN
    ; This should be one of the following operations
    ; CLRF CLRW
    IF_REG L=03 BEGIN
        ; Get contents of status register into R8 and Mask Flag Bits
        META IF_FOUND 2,DECOMP : BEGIN : DO 'CLRF' 'BH' : END
        ; Zero RA, Set ZERO Bit, Put contents of RA into W register
        META DO ADDO 0S048!8P ; SEL 3 ; CL 02 ; RA A
        RESTART FLAG_PROCESS
    END
    DO 0S7FL&BP0S80L&
    IF_REG 0=80 BEGIN
        ; Get contents of status register into R8 and Mask Flag Bits
        META SEL 2 : CL 03 : RD 8 ; DO 0S038&8P
        ; Zero RA, Set ZERO Bit, Put contents of RA into f register
        DO AS00 0S048!8P
        PERFORM PUT_REGFILE
        RESTART FLAG_PROCESS
    END
    END IF_REG H=02 BEGIN
    ; SUBWF Operation
    DO 0S7FL&BP0S80L&LP
    META IF_FOUND 2,DECOMP : BEGIN : DO 'SUBWF' : END
    ; Get contents of status register into R8 and Mask Flag Bits
    META SEL 3 : CL 02 : RD 8
    ; Process the Calculation leaving Result in RA
    DO CG2PAG5P
    PERFORM GENERAL_8_BIT_SUB
    DO 2GAP
    RESTART END_WF_OPERATION
    END IF_REG H=03 BEGIN
    ; DECF Operation
    DO 0S7FL&BP0S80L&LP
    META IF_FOUND 2,DECOMP : BEGIN : DO 'DECF' : END
    ; Get contents of status register into R8 and Mask Flag Bits
    META SEL 3 : CL 02 : RD 8
    ; Process the Calculation leaving Result in RA
    DO AGI-AP0SFFA&AP
    META IF_REG A=00 : BEGIN : DO 0S048!8P : END
    RESTART END_WF_OPERATION
    END IF_REG H=04 BEGIN
    ; IORWF Operation
    DO 0S7FL&BP0S80L&LP
    META IF_FOUND 2,DECOMP : BEGIN : DO 'IORWF' : END
    ; Get contents of status register into R8 and Mask Flag Bits
    META SEL 3 : CL 02 : RD 8
    ; Process the Calculation leaving Result in RA
    DO AGI-AP0SFFA&AP
    META IF_REG A=00 : BEGIN : DO 0S048!8P : END
    RESTART END_WF_OPERATION
    END IF_REG H=05 BEGIN
    ; ANDFWF Operation
    DO 0S7FL&BP0S80L&LP
    META IF_FOUND 2,DECOMP : BEGIN : DO 'ANDWF' : END
    ; Get contents of status register into R8 and Mask Flag Bits
    META SEL 3 : CL 02 : RD 8
    ; Process the Calculation leaving Result in RA
    DO AGI-AP0SFFA&AP
    META IF_REG A=00 : BEGIN : DO 0S048!8P : END
    RESTART END_WF_OPERATION
    END
; Process the Calculation leaving Result in RA
DO CGA&AP
META IF_REG A=00 : BEGIN : DO 0S048!8P : END
RESTART END_WF_OPERATION
END

IF_REG H=06
BEGIN
; XORWF Operation
; DO 0S7FL4BP0S80L&L
META IF_FOUND 2,DECOMP : BEGIN : DO 'XORWF' : END
; Get contents of status register into R8 and Mask Flag Bits
META SEL 2 : CL 03 : RD 8 : DO 0S038!8P
; Get contents of f register into RA
PERFORM GET_REGFILE
; Get contents of W register into RC
META SEL 3 : CL 02 : RD C
; Process the Calculation leaving Result in RA
DO CGAEAP
META IF_REG A=00 : BEGIN : DO 0S048!8P : END
RESTART END_WF_OPERATION
END

IF_REG H=07
BEGIN
; ADDWF Operation
; DO 0S7FL4BP0S80L&L
META IF_FOUND 2,DECOMP : BEGIN : DO 'ADDWF' : END
; Get contents of status register into R8 and Mask Flag Bits
META SEL 2 : CL 03 : RD 8 : DO 0S038!8P
; Get contents of f register into RA
PERFORM GET_REGFILE
; Process the Calculation leaving Result in RA
DO CGAEAP
PERFORM GENERAL_8_BIT_ADD
DO 2GAP
RESTART END_WF_OPERATION
END

RESTART GENERAL_FAIL
END

IF_REG_HIGHER H=07
BEGIN
IF_REG H=08
BEGIN
; MOVF Operation
; DO 0S7FL4BP0S80L&L
META IF_FOUND 2,DECOMP : BEGIN : DO 'MOVF' : END
; Get contents of status register into R8 and Mask Flag Bits
META SEL 2 : CL 03 : RD 8 : DO 0S038!8P
; Get contents of f register into RA
PERFORM GET_REGFILE
; Process the Calculation leaving Result in RA
META IF_REG A=00 : BEGIN : DO 0S048!8P : END
RESTART END_WF_OPERATION
END

IF_REG H=09
BEGIN
; COMF Operation
; DO 0S7FL4BP0S80L&L
META IF_FOUND 2,DECOMP : BEGIN : DO 'COMF' : END
; Get contents of status register into R8 and Mask Flag Bits
META SEL 2 : CL 03 : RD 8 : DO 0S038!8P
; Get contents of f register into RA
PERFORM GET_REGFILE
; Process the Calculation leaving Result in RA
DO CGAEAP
META IF_REG A=00 : BEGIN : DO 0S048!8P : END
RESTART END_WF_OPERATION
END

IF_REG H=0A
BEGIN
; INCF Operation
; DO 0S7FL4BP0S80L&L
META IF_FOUND 2,DECOMP : BEGIN : DO 'INCF' : END
; Get contents of status register into R8 and Mask Flag Bits
META SEL 2 : CL 03 : RD 8 : DO 0S038!8P
; Get contents of f register into RA
PERFORM GET_REGFILE

; Process the Calculation leaving Result in RA
DO AGI+AP0SFFA&AP

META IF_REG A=00 : BEGIN : DO O0S048!8P : END
RESTART END_WF_OPERATION

END

IF_REG H=0B
BEGIN

; DECFSZ Operation
DO TGI+TP 0S7FL&BP0S80L&LP

META IF_FOUND 2,DECOMP : BEGIN : DO 'DECFSZ' : END

; Get contents of f register into RA
PERFORM GET_REGFILE

; Process the Calculation leaving Result in RA
DO AGI-AP0SFFA&AP

; PC = PC + 1 WHEN RESULT = ZERO
IF_REG A=00 BEGIN
DO PGI+PP
IF_REG_HIGHER P,FF
META BEGIN : DO 0SFFP&PPQGI+QP0S1FQ&QP : END
END
PERFORM END_WF_OPERATION
RESTART GENERAL_EXIT
END

IF_REG H=0C
BEGIN

; RRF Operation
DO 0S7FL&BP0S80L&LP

META IF_FOUND 2,DECOMP : BEGIN : DO 'RRF' : END

; Get contents of status register into R8 and Mask Flag Bits
; and the C Bit in register RC
META SEL 2 : CL 03 : RD 8 : DO 0S018&CP0S068&8P

; Get contents of f register into RA
PERFORM GET_REGFILE

; Process the Calculation leaving Result in RA
DO 0S0A0A&AP AG0RAP 0SFEA&C!AP

PERFORM END_WF_OPERATION
RESTART GENERAL_EXIT
END

IF_REG H=0D
BEGIN

; RLF Operation
DO 0S7FL&BP0S80L&LP

META IF_FOUND 2,DECOMP : BEGIN : DO 'RLF' : END

; Get contents of status register into R8 and Mask Flag Bits
; and the C Bit in register RC
META SEL 2 : CL 03 : RD 8 : DO 0S018&CP0S068&8P

; Get contents of f register into RA
PERFORM GET_REGFILE

; Process the Calculation leaving Result in RA
DO 0S0A0A&AP AG0RAP 0SFEA4C!AP

PERFORM END_WF_OPERATION
RESTART GENERAL_EXIT

END

IF_REG H=0E
BEGIN

; SWAPF Operation
DO 0S7FL&BP0S80L&LP

META IF_FOUND 2,DECOMP : BEGIN : DO 'SWAPF' : END

; Get contents of status register into R8 and Mask Flag Bits
; and the C Bit in register RC
META SEL 2 : CL 03 : RD 8 : DO 0S018&CP0S068&8P

; Get contents of f register into RA
PERFORM GET_REGFILE

; Process the Calculation leaving Result in RA
DO 0S0A0A&AP AG0RAP 0SFEA4C!AP

PERFORM END_WF_OPERATION
RESTART GENERAL_EXIT

END

IF_REG H=0F
BEGIN

; INCFSZ Operation
DO TGI+TP 0S7FL&BP0S80L&LP

META IF_FOUND 2,DECOMP : BEGIN : DO 'INCFSZ' : END

; Get contents of f register into RA
PERFORM GET_REGFILE

; Process the Calculation leaving Result in RA
DO AGI+AP0SFFA&AP

; PC = PC + 1 WHEN RESULT = ZERO
IF_REG A=00
BEGIN
DO PGI+PP
IF_REG_HIGHER P,PF
META BEGIN : DO 0SFF4FPPQGI+QP0SIFQ&QP : END
END
PERFORM END_IF_OPERATION
END
END
RESTART GENERAL_EXIT

IF_REG 0=10
BEGIN
DO 0S3CH&
IF_REG 0=10
BEGIN
BCF Operation
; Register N = Bit , B = f
DO 0S7FP&BP 0S03H&B=N+NP 0S80L&
META IF_REG 0=80 : BEGIN : DO 'BCF 'BH','NH : END
META IF_FOUND 2,DECOMP : BEGIN : DO 0SFFP&PPQGI+QP0S1FQ&QP : END
; Get contents of f register into RA
PERFORM GET_REGFILE
; Process the Calculation leaving Result in RA
REG2BIT A
BIT_CLR N
BIT2REG A
PERFORM PUT_REGFILE
RESTART GENERAL_EXIT
END
END
IF_REG 0=14
BEGIN
BSF Operation
; Register N = Bit , B = f
DO 0S7FP&BP 0S03H&B=N+NP 0S80L&
META IF_REG 0=80 : BEGIN : DO 'BSF 'BH','NH : END
META IF_FOUND 2,DECOMP : BEGIN : DO 0SFFP&PPQGI+QP0S1FQ&QP : END
; Get contents of f register into RA
PERFORM GET_REGFILE
; Process the Calculation leaving Result in RA
REG2BIT A
BIT_SET N
BIT2REG A
PERFORM PUT_REGFILE
RESTART GENERAL_EXIT
END
END
IF_REG 0=18
BEGIN
BTFSC Operation
; Register N = Bit , B = f
DO TGI+TP 0S7FP&BP 0S03H&B=N+NP 0S80L&
META IF_REG 0=80 : BEGIN : DO 'BTFSC 'BH','NH : END
META IF_FOUND 2,DECOMP : BEGIN : DO 0SFFP&PPQGI+QP0S1FQ&QP : END
; Get contents of f register into RA
PERFORM GET_REGFILE
; Process the Calculation leaving Result in RA
BIT_LOAD 00000000
BIT_SET N
BIT2REG 0
DO N&
; PC = PC + 1 WHEN RESULT = ZERO
IF_REG 0=00
BEGIN
DO PGI+PP
IF_REG_HIGHER P,PF
META BEGIN : DO 0SFF4FPPQGI+QP0SIFQ&QP : END
END
RESTART GENERAL_EXIT
END
END
IF_REG 0=1C
BEGIN
BTFSS Operation
; Register N = Bit , B = f
DO TGI+TP 0S7FP&BP 0S03H&B=N+NP 0S80L&
META IF_REG 0=80 : BEGIN : DO 'BTFSS 'BH','NH : END
META IF_FOUND 2,DECOMP : BEGIN : DO 0SFFP&PPQGI+QP0S1FQ&QP : END
; Get contents of f register into RA
PERFORM GET_REGFILE
; Process the Calculation leaving Result in RA
BIT_LOAD 00000000
BIT_SET N
BIT2REG 0
DO N&
; PC = PC + 1 WHEN RESULT = ZERO
IF_REG NOT 0=00
BEGIN
DO PGI+PP
IF_REG_HIGHER P,PF
META BEGIN : DO 0SFF4FPPQGI+QP0SIFQ&QP : END
END
RESTART GENERAL_EXIT
END
END
IF_REG 0=1C
BEGIN
BTFSS Operation
; Register N = Bit , B = f
DO TGI+TP 0S7FP&BP 0S03H&B=N+NP 0S80L&
META IF_REG 0=80 : BEGIN : DO 'BTFSS 'BH','NH : END
META IF_FOUND 2,DECOMP : BEGIN : DO 0SFFP&PPQGI+QP0S1FQ&QP : END
; Get contents of f register into RA
PERFORM GET_REGFILE
; Process the Calculation leaving Result in RA
BIT_LOAD 00000000
BIT_SET N
BIT2REG 0
DO N&
; PC = PC + 1 WHEN RESULT = ZERO
IF_REG NOT 0=00
BEGIN
DO PGI+PP
IF_REG_HIGHER P,PF
META BEGIN : DO 0SFF4PPGSI+Q805IFQ4QP : END

END

END

META BEGIN : DO 0SFF4PPGSI+Q805IFQ4QP : END

RESTART GENERAL_EXIT

RESTART GENERAL_FAIL

IF_REG 0=20

BEGIN

; This should be one of the following operations

; CALL  GOTO

DO AGQPLGPPTGI+TP

RESTART GENERAL_EXIT

IF_REG 0=30

BEGIN

; This should be one of the following operations

ADDLW ANDLW IORLW XORLW RETLW MOVLW SUBLW

DO 0S3EH&

IF_REG 0=3E

BEGIN

META IF_FOUND 2,DECOMP : BEGIN : DO 'ADDLW 'LH : END

META SEL 3 : CL 02 : RD 2 : DO LG5P

PERFORM GENERAL_8_BIT_ADD

META DO 2GAP : CACHE_W A

RESTART FLAG_PROCESS

END

IF_REG 0=3C

BEGIN

META IF_FOUND 2,DECOMP : BEGIN : DO 'SUBLW 'LH : END

META SEL 3 : CL 02 : RD 2 : DO LG5P

PERFORM GENERAL_8_BIT_SUB

META DO 2GAP : CACHE_W A

RESTART FLAG_PROCESS

END

IF_REG 0=33

BEGIN

META IF_FOUND 2,DECOMP : BEGIN : DO 'MOVLW 'LH : END

META SEL 3 : CL 02 : WR L

RESTART GENERAL_EXIT

END

IF_REG 0=34

BEGIN

META IF_FOUND 2,DECOMP : BEGIN : DO 'RETLW 'LH : END

META SEL 3 : CL 02 : WR L

DO TGI+TP

PERFORM POP

RESTART GENERAL_EXIT

END

IF_REG H=38

BEGIN

META IF_FOUND 2,DECOMP : BEGIN : DO 'IORLW 'LH : END

META SEL 2 : CL 03 : RD 8 : DO 0S038&8P

META SEL 3 : CL 02 : RD 2 : DO LG5P

META DO LG5AP : CACHE_W A

DO LG5AP

META IF_REG A=00 : BEGIN : DO 0S048!8P : END

CACHE_W A

RESTART FLAG_PROCESS

END

IF_REG H=39

BEGIN

META IF_FOUND 2,DECOMP : BEGIN : DO 'XORLW 'LH : END

META SEL 2 : CL 03 : RD 8 : DO 0S038&8P

META SEL 3 : CL 02 : RD 2 : DO LG5P

META DO LG5AP : CACHE_W A

DO LG5AP

META IF_REG A=00 : BEGIN : DO 0S048!8P : END

CACHE_W A

RESTART FLAG_PROCESS

END

IF_REG H=3A

BEGIN

META IF_FOUND 2,DECOMP : BEGIN : DO 'XORLW 'LH : END

META SEL 2 : CL 03 : RD 8 : DO 0S038&8P

META SEL 3 : CL 02 : RD 2 : DO LG5P

META DO LG5AP : CACHE_W A

DO LG5AP

META IF_REG A=00 : BEGIN : DO 0S048!8P : END

CACHE_W A

RESTART FLAG_PROCESS

END

IF_REG H=3B

BEGIN

META IF_FOUND 2,DECOMP : BEGIN : DO 'ADDLW 'LH : END

META SEL 3 : CL 02 : RD 2 : DO LG5P

PERFORM GENERAL_8_BIT_ADD

META DO 2GAP : CACHE_W A

RESTART FLAG_PROCESS

END

IF_REG H=3C

BEGIN

META IF_FOUND 2,DECOMP : BEGIN : DO 'SUBLW 'LH : END

META SEL 3 : CL 02 : RD 2 : DO LG5P

PERFORM GENERAL_8_BIT_SUB

META DO 2GAP : CACHE_W A

RESTART FLAG_PROCESS

END

IF_REG H=3D

BEGIN

META IF_FOUND 2,DECOMP : BEGIN : DO 'MOVLW 'LH : END

META SEL 3 : CL 02 : WR L

RESTART GENERAL_EXIT

END

IF_REG H=3E

BEGIN

META IF_FOUND 2,DECOMP : BEGIN : DO 'RETLW 'LH : END

META SEL 3 : CL 02 : WR L

DO TGI+TP

PERFORM POP

RESTART GENERAL_EXIT

END
657    ;       ******************************************************************
658    ;       *   SPECIAL CASE COMMON EXIT ROUTINES
659    ;       ******************************** **********************************
660    ;       Register A = Result             Register 8 = Flag Word
661    ;       Register L = f or W indicator
662    ;       RESTART FLAG_PROCESS
663    ;       END
664
665    ;       ******************************************************************
666    ;       *   GENERAL EXIT for end of WF Update Processes
667    ;       ******************************** **********************************
668    ;   Register A = Result             Register 8 = Flag Word
669    ;   Register L = f or W indicator
670    ;       BEGIN END_WF_OPERATION
671    ;       PERFORM END_WF_OPERATION
672    ;       RESTART FLAG_PROCESS
673    ;       END   END_WF_OPERATION
674
675    ;       ******************************************************************
676    ;       *   GENERAL EXIT with ALU Update Process
677    ;       ******************************** **********************************
678    ;   Register 2 = Acumulator         Register 8 = Flag Word
679    ;       BEGIN ALU_PROCESS
680    ;       CACHE_SEL 1
681    ;       CACHE_LOAD 00
682    ;       CACHE_R A
683    ;       DO 0S078&8P0SF8A&8!
684    ;       CACHE_W 0
685    ;       Next Instruction Not Needed as required Exit Code follows Immediately
686    ;       RESTART GENERAL_EXIT
687    ;       END ALU_PROCESS
688
689    ;       ******************************************************************
690    ;       *   GENERAL EXIT with FLAG Update only Process
691    ;       ******************************** **********************************
692    ;       Register 8 = Flag Word
693    ;       BEGIN FLAG_PROCESS
694    ;       CACHE_SEL 2
695    ;       CACHE_LOAD 03
696    ;       CACHE_R A
697    ;       DO 805905P805P8A18!
698    ;       CACHE_W 0
699    ;       Next Instruction Not Needed as required Exit Code follows Immediately
700    ;       RESTART GENERAL_EXIT
701    ;       END FLAG_PROCESS
702
703    ;       ******************************************************************
704    ;       *   GENERAL EXIT
705    ;       ******************************** **********************************
706    ;       BEGIN GENERAL_EXIT
707    ;       THIS IS THE COMMON EXIT PATH FOR MOST CODE
708    ;       PERFORM RETURN_PC
709    ;       META IF_FOUND 2,DECOMP : BEGIN : TRACE_M : END
710    ;       EXIT_REG T,OK
711    ;       END GENERAL_EXIT
712
713    ;       ******************************************************************
714    ;       *   MANUAL COMMAND TEST SECTION
715    ;       ******************************** **********************************
716    ;       IF_PARM 1=CONTROL
717    ;       BEGIN
718    ;       IF_PARM 2=TESTER
719    ;       BEGIN
720    ;       TRACE *********** TESTER **************
721    ;       DO AS01NS00
722    ;       PERFORM PUT_BIT
723    ;       DO 2S155S10
724    ;       PERFORM GENERAL_8_BIT_ADD
725    ;       DO 2S175SC2
726    ;       PERFORM GENERAL_8_BIT_ADD
727    ;       DO 2S025S01
PERFORM GENERAL_8_BIT_SUB
DO 252502
PERFORM GENERAL_8_BIT_SUB
DO 252503
PERFORM GENERAL_8_BIT_SUB
END

;******************************************************************************
;       *   PROCEDURES SECTION
;       ******************************** **********************************

PROCEDURE XXX
; Entry Conditions
;      Register 0 is
; ; Exit Conditions
;      ...

END_PROCEDURE XXX

PROCEDURE GET_REGFILE
; Entry Conditions
;       Register B is Register to access 0 -> 7F
; ; Exit Conditions
;       Register A is Contents of Accessed Register
;       <STATUS> Register will hold High bit/s as appropriate to
;           adapt the value supplied in B Register
;           The full address is calculated into Register pair B & C
; CACHE_SEL 2
CACHE_LOAD 03
;Register S = <STATUS> Bit 7 = IRP , Bit 6 & 5 = RP1 & RP0
CACHE_R S
META DO 0S7FCS00BP05040S4 : IF_REG 0=40 : BEGIN : DO CS01 : END
META DO 0S20S4 : IF_REG 0=20 : BEGIN : DO 0S80BP : END
META DO 0STFB4 : IF_REG 0=00 BEGIN
; This is an Indirect Operation Memory Access process
DO 0S80S4
META_IF_REG 0=80 : BEGIN : DO CS01 : END
CACHE_LOAD 04
CACHE_R B
DO 0STFB4
; IF INDIRECT REGISTER RESELECTED AGAIN WRITE = NOP , READ = 00H
META_IF_REG 0=00 : BEGIN : DO AS00 : RETURN : END
DO 0STFB4
END

BIT{2REG B
CACHE_BITLOAD
CACHE_R A
END_PROCEDURE GET_REGFILE

PROCEDURE PUT_REGFILE
; Entry Conditions
;       Register A is New Contents of Accessed Register
;       Register B is Register to access 0 -> 7F
; ; Exit Conditions
; ; Note <STATUS> Register will hold High bit/s as appropriate to
;           adapt the value supplied in B Register
;           The full address is calculated into Register pair B & C
; CACHE_SEL 2
CACHE_LOAD 03
;Register S = <STATUS> Bit 7 = IRP , Bit 6 & 5 = RP1 & RP0
CACHE_R S
META DO 0STFC00BP05040S4 : IF_REG 0=40 : BEGIN : DO CS01 : END
META_IF_REG 0=40 BEGIN
CACHE_SEL 1
CACHE_LOAD 08
CACHE_R 0
DO CS01110C
CACHE_SEL 2
END
META DO 0S20S4 : IF_REG 0=20 : BEGIN : DO 0S80BP : END
META DO 0STFB4 : IF_REG 0=00 BEGIN
; This is an Indirect Operation Memory Access process
DO 0S80S4
META_IF_REG 0=80 : BEGIN : DO CS01 : END
CACHE_LOAD 04
CACHE_R B
DO 0STFB4
; IF INDIRECT REGISTER RESELECTED AGAIN WRITE = NOP , READ = 00H
META_IF_REG 0=00 : BEGIN : DO AS00 : RETURN : END
DO 0STFB4
IF_REG B=03
END

IF_REG 0=02
BEGIN
; This is a PC Low Byte write process hence Load High Byte with PCLATH
CACHE_LOAD 0A
CACHE_R Q
DO 0S1FQER
IF_REG B=03
BEGIN
; This is a status register process
DO DDATAAP
CACHE_LOAD 03
CACHE_W A
RETURN

; BIT(2REG 8)
CACHE_BITLOAD
CACHE_W A

; PROCEDURE PUT_REGFILE

; PROCEDURE GENERAL_8_BIT_ADD
; Entry Conditions
; R2 = BYTE 1         R5 = BYTE 2
; Exit Conditions
; R8 = FLAG WORD (Status Bits)
; R2 = BYTE 1 + BYTE 2
;
; Flag Word Bit Allocation Usage (PIC)
; Bit     7   6   5   4   3   2   1   0
; Use     *   *   *   *   *   Z   DC  C
;
; Register Usage
; R1 = SIGN CARRY IN   R2 = HIGH CHUNK 1
; R3 = LOW CHUNK 1     R5 = HIGH CHUNK 2
; R6 = LOW CHUNK 2     R7 = CARRY IN   R8 = SIGN BIT
;
; Calculate Sum of Low LS 4 Bits Section , Set Sign Carry In as reqd
; R7 = R3+R6 (ie. Sum Value of LS 4 Bits) ... R1 = Sign Carry OUT
;
; TRACE ****************************
; TRACE BEFORE 8 BIT ADD
;
; Calculate Half Carry Out Section
; R2 = (Sign + LS Chunk)
;
; TRACE ****************************
; TRACE BEFORE 8 BIT SUB
;
; META IF_REG 2=00 , BEGIN : DO 05048!8P : TRACE RESULT IS ZERO : END

; PROCEDURE GENERAL_8_BIT_SUB
; Entry Conditions
; R2 = BYTE 1         R5 = BYTE 2
; Exit Conditions
; R8 = FLAG WORD
; R2 = BYTE 1 - BYTE 2
;
; Flag Word Bit Allocation Usage (PIC)
; Bit     7   6   5   4   3   2   1   0
; Use     CY                 OVR     PAR
;
; Register Usage
; R0 = ...   R1 = SIGN CARRY IN   R2 = HIGH CHUNK 1
; R3 = LOW CHUNK 1     R5 = HIGH CHUNK 2
; R6 = LOW CHUNK 2     R7 = CARRY IN   R8 = SIGN BIT
;
; TRACE ****************************
; TRACE BEFORE 8 BIT SUB
;
; META IF_REG 2=00 , BEGIN : DO 05048!8P : TRACE RESULT IS ZERO : END

; PROCEDURE GET_OP_CODE
; Set Time State Transitions Counter + 1T
; Select Internal Area
; Get Address of the PIC PC & Access Locally
;
; Get ISPAC Cache Pointer NOTE PC Address Doubled as 2 Byte per Opcode
CACHE_SEL 1
BIT|REG P
CACHE_BITLOAD
CACHE_BITADD
CACHE_R H
;Strip Binary to 14 Bits No matter what is loaded in ISPACE
DO 0S3FH&HP
CACHE_ADD I
CACHE_R L
; IF_FOUND 2,DEBUG
BEGIN
P, Q = PC , H,L=OPCODE
TRACE_R Q  "PC HIGH ="
TRACE_R P  "PC LOW  ="
TRACE_R H  "(PCH)    ="
TRACE_R L  "(PCL)    ="
END
META IF_FOUND 2,PC : BEGIN : DO 'PC='QHPH'  (PC)='HHLH'  ' : END
PC = PC + 1
DO PGI+PP
META IF_REG_HIGHER P,FF : BEGIN : DO  0SFFP&PPQGI+QP0S1FQ&QP : END
END_PROCEDURE GET_OP_CODE
PROCEDURE RETURN_PC
ENTRY CONDITIONS
RQ = PCH        RP = PCL        Location 07 = Address High Byte Mask
EXIT CONDITIONS
PC Stored in Internals Area
CACHE_SEL 3
CACHE_LOAD 07
CACHE_R 0
DO Q&QP
CACHE_LOAD 00
CACHE_W Q
CACHE_LOAD 01
CACHE_W P
END_PROCEDURE RETURN_PC
PROCEDURE PUSH
ENTRY Conditions
Register P is PCH
Register Q is PCL
EXIT Conditions
P + Q placed on Stack and Stack Pointer Adjusted
Initially locate SP Address , Limit Range , Write PCH,PCL on stack
CACHE_SEL 3
CACHE_LOAD 06
CACHE_R 1
DO 0S0F1&1P
CACHE_W 1
CACHE_LOAD 10
CACHE_ADD I
CACHE_W P
CACHE_ADD I
CACHE_W Q
; Update SP Address ie SP=SP+2 , Limit Range , Write updated SP back
DO 0S021+1P0S0F1&1P
CACHE_LOAD 06
CACHE_W 1
END_PROCEDURE PUSH
PROCEDURE POP
ENTRY Conditions
P + Q Removed from Stack and Stack Pointer Adjusted
Initially locate SP Address , Update SP Address ie SP=SP-2
Limit Range , Write updated SP back , Read PCH,PCL from stack
CACHE_SEL 3
CACHE_LOAD 06
CACHE_R 1
DO 1GI-I-1P0S0F1&1P
CACHE_W 1
CACHE_LOAD 10
CACHE_R P
CACHE_ADD I
CACHE_R Q
END_PROCEDURE POP
PROCEDURE END_W&F_OPERATION
ENTRY Conditions
Register A = Result
Register L = f or W indicator
EXIT Conditions
Decomp Processed and Location Written to
META IF_FOUND 2,DECOMP : BEGIN : DO ' 'BH
META IF_REG L=00 : BEGIN : DO ' ,0' : END
META IF_REG L=80 : BEGIN : DO '1' : END : END
META IF_REG L=80 : BEGIN : PERFORM PUT_REGFILE : END
META IF_REG L=00 : BEGIN : SEL 3 : CL 02 : MR A : END
END_PROCEDURE END_W&F_OPERATION
******************************** **********************************
IF_PARM 1=CONTROL
BEGIN
  TRACE Control Command Located
  EXIT_CODE OK
END

IF_PARM 1=DEVICE
BEGIN
  IF_PARM 2=FLASHER
  BEGIN
    DEVICE R02,C20,'S BELOW'
    DEVICE_REG S R03,C20 ;Write S to Screen
    IF_REG S=0
    BEGIN
      DO SS01
      DEVICE R07,C30,<"Flash ON ">
      EXIT_CODE OK
    END
    IF_REG S=1
    BEGIN
      DO SS02
      DEVICE R07,C30,<'Flash OFF'>
      EXIT_CODE OK
    END
    IF_REG S=2
    BEGIN
      DO SS03
      DEVICE R07,C30,"Flash ON "
      EXIT_CODE OK
    END
    IF_REG S=3
    BEGIN
      DO SS00
      DEVICE R07,C30,'Flash OFF'
      EXIT_CODE OK
    END
    EXIT_CODE FAIL
  END
  EXIT_CODE FAIL
END

IF_PARM 1=PROCESSOR_RESET
BEGIN
  CACHE_MAP 1 ISPACE
  CACHE_MAP 2 REGISTERS
  CACHE_MAP 3 INTERNALS
  ;Select a Specific Cache Map (INTERNALS AREA)
  CACHE_SEL 3
  ;Reset Time State Counter
  DO TS00
  ;Clear PCH=0 & PCL=0
  CACHE_LOAD 00
  CACHE_W T
  ;Clear Stack Pointer & its contents
  CACHE_LOAD 01
  CACHE_W T
  DO TS03
  ;Initialize PC High Mask (Default = 1K)
  ;Initialize FSR's High Max Mask (Default = 2 Banks)
  DO TS03
END

IF_PARM 1=PROCESSOR_RESET
BEGIN
  CACHE_MAP 1 ISPACE
  CACHE_MAP 2 REGISTERS
  CACHE_MAP 3 INTERNALS
  ;Select a Specific Cache Map (REGISTERS AREA)
CACHE_SEL 2 ;Reset STATUS
DO TS18
CACHE_LOAD 03 CACHE_M T
;Reset PC/LATH INTCON
DO TS00
CACHE_LOAD 0A CACHE_M T
CACHE_LOAD 0B CACHE_M T
;Reset OPTION , TRISA , TRISB
DO TS00
CACHE_LOAD 08
CACHE_W T
CACHE_LOAD 09
CACHE_W T
CACHE_LOAD 0A
CACHE_W T
CACHE_LOAD 0B
CACHE_W T
;Reset EECON1 , TRISA , TRISB
DO TSFF
CACHE_LOAD 81
CACHE_W T
CACHE_LOAD 85
CACHE_W T
CACHE_LOAD 86
CACHE_W T
CACHE_LOAD 88
CACHE_W T
EXIT_REG T,OK
END

; *****************************************************
;                    DEVICE INIT IALISE
; *****************************************************
IF_PARM 1=DEVICE_RESET BEGIN
IF_PARM 2=FLASHER BEGIN
DO SS00
DEVICE R99
DEVICE R01,C01,'*
DEVICE R15,C01,'*
DEVICE R01,C79,'*
DEVICE R15,C79,'*
CACHE_MAP_REV 7 I/O
EXIT_CODE OK
END
EXIT_CODE FAIL
END

; *****************************************************
;                    SYSTEM INIT I ALISE
; *****************************************************
IF_PARM 1=~~LIB_CHECK BEGIN
META IF_SIMULATOR 16C84 : BEGIN : EXIT_CODE OK : END
META EXIT_CODE Failed
END

; *****************************************************
;                    HARDWARE IN ITIALISE
; *****************************************************
IF_PARM 1=~~CONFIG BEGIN
META IF_PARM 2=00 : BEGIN : EXIT_CODE .TYPE INTERNALS : END
META IF_PARM 2=01 : BEGIN : EXIT_CODE .PHYSICAL RO# 40 INTERNALS : END
META IF_PARM 2=02 : BEGIN : EXIT_CODE END : END
META IF_PARM 2=03 : BEGIN
META EXIT_CODE .MODIFY 01 00 00 X RW % <PCH>
META IF_PARM 2=04 : BEGIN
META EXIT_CODE .MODIFY 01 01 01 X RW % <PCL>
META IF_PARM 2=05 : BEGIN
META EXIT_CODE .MODIFY 01 02 02 RW R W% <W>
META EXIT_CODE .MODIFY 01 02 02 X R W% <W>
META IF_PARM 2=06 : BEGIN
META EXIT_CODE .MODIFY 01 03 03 X RW % <Bus Byte>
META IF_PARM 2=09 : BEGIN
META EXIT_CODE .MODIFY 01 05 05 X RW % <Addr Lo>
META IF_PARM 2=45 : BEGIN
META EXIT_CODE .MAPPED 01 100 100 07F <Register Banks 2,3> : END
META IF_PARM 2=46 : BEGIN : EXIT_CODE .MAPPED END : END
META IF_PARM 2=47 : BEGIN : EXIT_CODE .TYPE VECTORS : END
META IF_PARM 2=48 : BEGIN : EXIT_CODE .PHYSICAL RO# 10 VECTORS : END
META IF_PARM 2=49 : BEGIN : EXIT_CODE .PHYSICAL END : END
META IF_PARM 2=4A : BEGIN
META EXIT_CODE .MODIFY 01 02 06 X UD - <Undefined> : END
META IF_PARM 2=4B : BEGIN : EXIT_CODE .MODIFY END : END
META IF_PARM 2=4C : BEGIN
META EXIT_CODE .MAPPED 01 00 00 10 <Vector Map 1> : END
META IF_PARM 2=4D : BEGIN : EXIT_CODE .MAPPED END : END
META IF_PARM 2=4E : BEGIN : EXIT_CODE .TYPE ISPACE 2 : END
META IF_PARM 2=4F : BEGIN : EXIT_CODE .PHYSICAL RO# 1000 ISPACE : END
META IF_PARM 2=50 : BEGIN : EXIT_CODE .PHYSICAL END : END
META IF_PARM 2=51 : BEGIN
META EXIT_CODE .MAPPED 01 00 00 1000 <14 Bit Instructions> : END
META IF_PARM 2=52 : BEGIN : EXIT_CODE .MAPPED END : END
META IF_PARM 2=53 : BEGIN : EXIT_CODE .TYPE END_PROCESSOR_CONFIG : END
META IF_PARM 2=COUNT : BEGIN : EXIT_CODE 53 : END
END

START<decimal> DEVICE_CONFIG
BEGIN
META IF_PARM 2=00 : BEGIN : EXIT_CODE .TYPE EXTERNAL : END
META IF_PARM 2=01 : BEGIN : EXIT_CODE .PHYSICAL RO# 20 EXTERNAL : END
META IF_PARM 2=02 : BEGIN : EXIT_CODE .PHYSICAL END : END
META IF_PARM 2=03 : BEGIN
META EXIT_CODE .MODIFY 01 00 00 X RW - <Display 1 Segments> : END
META IF_PARM 2=04 : BEGIN
META EXIT_CODE .MODIFY 01 01 01 X RW - <Display 2 Segments> : END
META IF_PARM 2=05 : BEGIN
META EXIT_CODE .MODIFY 01 02 02 X RW - <Display 3 Segments> : END
META IF_PARM 2=06 : BEGIN
META EXIT_CODE .MODIFY 01 03 03 X RW - <Display 4 Segments> : END
META IF_PARM 2=07 : BEGIN
META EXIT_CODE .MODIFY 01 04 04 X RW - <Display 1 Bright Level> : END
META IF_PARM 2=08 : BEGIN
META EXIT_CODE .MODIFY 01 05 05 X RW - <Display 2 Bright Level> : END
META IF_PARM 2=09 : BEGIN
META EXIT_CODE .MODIFY 01 06 06 X RW - <Display 3 Bright Level> : END
META IF_PARM 2=0A : BEGIN
META EXIT_CODE .MODIFY 01 07 07 X RW - <Display 4 Bright Level> : END
META IF_PARM 2=0B : BEGIN
META EXIT_CODE .MODIFY 01 08 08 X RO# <Unused> : END
META IF_PARM 2=0C : BEGIN
META EXIT_CODE .MAPPED 01 00 00 20 <4 * 7 Segment Map> : END
META IF_PARM 2=0D : BEGIN : EXIT_CODE .MAPPED END : END
META IF_PARM 2=0E : BEGIN : EXIT_CODE .TYPE ENDDEVICE_CONFIG : END
META IF_PARM 2=COUNT : BEGIN : EXIT_CODE 0E : END
END

RESTART<decimal> LIBRARY_END

;**********************************************************************
; Library Linking Section
;**********************************************************************
BEGIN<decimal> LIBRARY_END

; This is the Library Extention Linking Point (A dummy access point)
END<decimal> LIBRARY_END

;**********************************************************************
; End of Library
;**********************************************************************

;
This document was created on 23/August/2011 as reproduction of the original report created by May 2002 and submitted as part of The Applied Computing and Electronic BEng Hons) Course held at Bournemouth University.

Some entries may have adjusted dates due to this documents recent production and also slightly differing page lengths as a result of using alternative development software. for print production.