Contents

Abbreviations **xvii**

Introduction **xix**

Acknowledgements **xxiii**

1 Introducing embedded systems and the microcontroller **1**

1.1 Embedded systems and their characteristics 2
1.1.1 The essence of the embedded system 2
1.1.2 Further features of the embedded system 5
1.1.3 The skills of the embedded system designer 7
1.2 Our starting point: the microprocessor 7
1.2.1 The microprocessor reviewed 7
1.2.2 More on instructions, and the ALU 11
1.3 Some microprocessor design options 13
1.3.1 Von Neumann and Harvard 13
1.3.2 Instruction sets – CISC and RISC 14
1.3.3 Instruction pipelining 15
1.4 The microcontroller: its applications and environment 16
1.4.1 Microcontroller characteristics 17
1.4.2 Features of a general-purpose microcontroller 18
1.4.3 Some example controllers 18
1.5 Microchip Inc. and the PIC microcontroller 19
1.5.1 Background, and meet the family 19
1.5.2 A 16F84 microcontroller overview 20

1.5.3 CPU, programming model and instruction set 22
1.6 The Philips 80C552 microcontroller 22
1.6.1 Background, and meet the family 22
1.6.2 Controller overview and architecture 23
1.6.3 CPU, programming model and instruction set 23
1.7 The Motorola 68HC05/08 microcontrollers 25
1.7.1 Background, and meet the family 25
1.7.2 68HC05 controller overview 26
1.7.3 CPU, programming model and instruction set 27
1.7.4 The 68HC08 28

2 From humble beginnings – towards the minimum system 32
2.1 Interfacing with peripherals 33
2.1.1 Special Function Registers 33
2.2 Interrupts 34
2.2.1 An interrupt review 34
2.2.2 Interrupt hierarchy 36
2.2.3 The 16F84 interrupt structure 37
2.3 Parallel input/output (I/O) ports 38
2.3.1 The bidirectional port 38
2.3.2 The 16F84 ports 39
2.3.3 Interrupt on change 40
2.3.4 Unassigned inputs and weak pull-ups 41
2.3.5 Electrical characteristics 41
2.3.6 The quasi-bidirectional port 42
2.4 Simple interfacing 43
2.4.1 Switches 43
2.4.2 Light-emitting diodes 45
2.5 Counters and timers 47
2.5.1 The digital counter reviewed 47
2.5.2 The 16F84 TIMER0 module 48
2.5.3 Applications of the Counter/Timer 50
2.5.4 A Counter/Timer enhancement: the auto-reload 52
2.5.5 The Watchdog Timer 53
2.6 Power supply and reset 54
2.6.1 Power supply requirements 54
2.6.2 Reset 54
2.6.3 Power-up 55

2.7 The clock oscillator 57
2.7.1 *R–C* oscillators 58
2.7.2 The quartz crystal 59
2.7.3 The ceramic resonator 60
2.7.4 The 16F84 oscillator 61
2.8 Some practical build tips 61

3 Preliminary programming 65
3.1 An introduction to Assembler 66
3.2 Prerequisites for programming 67
3.2.1 Memory maps 67
3.2.2 16F84 instruction format and addressing modes 68
3.2.3 Exploring the 16F84 instruction set by function 73
3.3 Writing in Assembler 74
3.3.1 Source code format 74
3.3.2 Assembler file structure 76
3.3.3 Assembler directives 78
3.4 Developing the program 78
3.4.1 Flow diagrams 78
3.4.2 Program flow 79
3.4.3 Starting the program 80
3.4.4 Subroutines 82
3.4.5 Interrupt service routines (ISRs) 83
3.4.6 Macros 84
3.4.7 Laying out the program 84
3.5 Generating time delays and intervals 86
3.5.1 Software-generated delays 86
3.5.2 Hardware-generated delays 87
3.5.3 Initiating repetitive time-based activity 88
3.6 Data handling 89
3.6.1 Data types 89
3.6.2 Tables and strings 90
3.6.3 The stack 92
3.6.4 Queues and buffers 93
3.7 Commissioning the minimum system 94
3.7.1 The Instruction Set Simulator 94
3.7.2 Program-related faults 96
3.7.3 Program download: connecting hardware and software 97
3.7.4 Hardware commissioning 97

4 Memory matters **102**
4.1 A memory overview 103
4.1.1 The memory array 105
4.2 Memory technologies 1: volatile memory 106
4.2.1 Dynamic RAM (DRAM) 106
4.2.2 Static RAM (SRAM) 107
4.3 Memory technologies 2: non-volatile memory 108
4.3.1 EPROM 109
4.3.2 One time programmable (OTP) 110
4.3.3 EEPROM 110
4.3.4 Flash 112
4.4 Microcontroller memory implementation 113
4.4.1 Memory implementation – on-chip or off? 113
4.4.2 Program memory 115
4.4.3 16F84 Memory programming 116
4.5 External memory devices 118
4.5.1 EPROM: the AMD (Advanced Micro Devices) 27C128 119
4.5.2 Static RAM: the Hitachi HM6264 121
4.5.3 EEPROM: Xicor X24165 121
4.5.4 Flash: the Atmel AT29C010A 123

5 Analogue affairs **127**
5.1 Digital to analogue (D to A) conversion 128
5.1.1 The standalone converter 129
5.1.2 Pulse width modulation (PWM) 130
5.1.3 Generating PWM signals in software 132
5.1.4 Generating PWM signals in hardware 133
5.2 Data acquisition systems 136
5.2.1 Acquiring AC signals: sampling rate and aliasing 136
5.2.2 Signal conditioning 137
5.2.3 The analogue multiplexer 139
5.2.4 Filtering 139
5.2.5 The Sample and Hold 142
5.3 Principles of analogue to digital conversion 144
5.3.1 Interpreting specifications 145
5.3.2 The voltage reference 147
5.3.3 The successive approximation ADC 149
5.3.4 The switched capacitor successive approximation ADC 151
5.4 A to D conversion in the microcontroller environment 153
5.4.1 Where there is no ADC 153

5.4.2 ADCs in microcontrollers 154
5.5 Selecting an ADC 160

6 Strictly serial 166
6.1 Serial communication overview 167
6.1.1 Synchronous data communication 167
6.1.2 Asynchronous data communication 168
6.1.3 So which do we choose? 169
6.1.4 Some terminology 169
6.2 Physical limitations to serial transmission 169
6.2.1 The short data link 170
6.2.2 Transmission line effects 171
6.2.3 Electromagnetic interference (EMI) 172
6.2.4 Ground differentials 173
6.3 Serial standards and protocols 173
6.3.1 Exploring protocols 173
6.3.2 Microwire™ and SPI (Serial Peripheral Interface) 175
6.3.3 The Inter-Integrated Circuit (I^2C) bus 177
6.3.4 RS-232/EIA-232 181
6.3.5 Controller Area Network (CAN) bus 183
6.4 Serial ports 186
6.4.1 Simple synchronous serial ports 186
6.4.2 A complex synchronous port: the I^2C port 187
6.4.3 Asynchronous serial ports 188
6.4.4 'Bit banging' 190

7 Systematic software 196
7.1 Overviewing software development 197
7.1.1 The software life cycle 197
7.1.2 The aims of software development 198
7.1.3 The process of software development 199
7.2 Developing program structure 200
7.2.1 Program flow 200
7.2.2 Modules 202
7.2.3 Taming the flow diagram – the module flow diagram 202
7.2.4 Top-down decomposition: the structure diagram 203
7.3 Assembler or high-level language? 206
7.3.1 Assemblers, compilers and interpreters 207
7.3.2 Assembler evaluated 207
7.3.3 A high-level language survey 208

7.4 The C programming language – an overview 209
7.4.1 Data types 210
7.4.2 C functions 210
7.4.3 Keywords 211
7.4.4 The C pre-preprocessor and pre-preprocessor directives 211
7.4.5 Program layout 212
7.4.6 C operators 213
7.4.7 Flow control 213
7.4.8 Files and file structure 214
7.5 Using C in the embedded environment 214
7.5.1 The MPLAB™-CXX compilers 216
7.5.2 Portability issues 217
7.5.3 Safety-conscious C – MISRA C 218
7.6 Optimising the development process 218
7.6.1 Version control 219
7.6.2 Style and layout 220
7.6.3 Learning from experience 220

8 Dealing with time 223
8.1 What is 'real-time'? 224
8.2 Advanced interrupt structures 225
8.2.1 Prioritisation 226
8.2.2 Interrupt latency 227
8.2.3 Interrupts without tears 232
8.2.4 Conclusions on interrupts 233
8.3 Advanced Counter/Timer (C/T) structures and applications 233
8.3.1 Time measurement and Counter/Timer specification 233
8.3.2 Frequency measurement 236
8.3.3 Counter/Timer enhancements 236
8.3.4 Device example: the 80C552 Timer 2 238
8.3.5 The Watchdog Timer (WDT) 240
8.4 Multi-tasking 241
8.5 Real-Time Operating Systems 245
8.5.1 Scheduling and the scheduler 246
8.5.2 Tasks 247
8.5.3 Data and resource protection 248
8.5.4 Implementing the RTOS 248

9 Interfacing to external devices 252
9.1 More on the digital interface 253
9.1.1 Interfacing digital input signals 253

9.1.2 Switch debouncing 254
9.1.3 Switch arrays and keypads 257
9.1.4 Transistor switches 259
9.2 Optical devices 1: displays 261
9.2.1 The seven-segment LED display 262
9.2.2 The liquid crystal display (LCD) 263
9.2.3 Hitachi LCD modules and the HD44780 microcontroller 265
9.3 Optical devices 2: simple opto-sensors 268
9.3.1 The opto-isolator 269
9.3.2 Object sensors 270
9.3.3 Shaft encoders 271
9.4 Inductive loads 273
9.4.1 On/off switching 273
9.4.2 Relays 275
9.4.3 Reversible switching – the H-bridge 276
9.4.4 PWM revisited – continuously variable control 277
9.5 The DC motor 278
9.5.1 Motor principles 278
9.5.2 Driving the DC motor 281
9.6 Stepping motors 282
9.6.1 Introducing the stepping motor 282
9.6.2 Drive waveforms 285
9.6.3 Motor speed profiles 286
9.6.4 Switching the motor phases 287
9.6.5 The motor drive system 288

10 Supplying and using power in a power-conscious world 293
10.1 Sources of power 294
10.1.1 The problem of power 294
10.1.2 Batteries 296
10.2 Supply voltage control 298
10.2.1 Power conditioning 298
10.2.2 The voltage regulator 298
10.2.3 The linear voltage regulator 299
10.2.4 Switching converters 302
10.3 Power supply supervision 307
10.4 CMOS – the low-power technology 310
10.5 The low-power microcontroller 313
10.6 Low-power circuit design 314

11 Dealing with numbers **321**
11.1 Fixed point number representation 322
11.1.1 Fractional binary numbers 322
11.1.2 Integer and fixed point arithmetic 323
11.2 Fixed point numerical routines 323
11.2.1 Addition and subtraction 324
11.2.2 Multiplication 326
11.2.3 Multi-byte multiplication using 8-bit hardware multiplier 327
11.2.4 Division 328
11.2.5 Conversion between binary and BCD 329
11.3 Floating point number representation 335
11.4 Truncation and rounding 337
11.5 Operations on sampled data 338
11.5.1 Averaging 338
11.5.2 Differentiation 341
11.5.3 Prediction 344
11.5.4 Integration 345

12 Designing and commissioning the system **349**
12.1 The engineering design process 350
12.1.1 A definition 350
12.1.2 The design process 350
12.1.3 Variant design 352
12.2 The design model applied to the embedded system 352
12.2.1 Defining the problem: specification development 353
12.2.2 Conceptual design 356
12.2.3 Embodiment design 358
12.3 Detail design 365
12.4 Commissioning the system 365
12.4.1 Commissioning strategies 365
12.4.2 The test procedure 368
12.5 Further diagnostic tools and their use 369
12.5.1 ROM/EPROM emulator 369
12.5.2 Monitor 370
12.5.3 Background Debug Mode (BDM) 370
12.5.4 In-Circuit Emulator (ICE) 371
12.5.5 The logic analyser 372
12.6 Some trends in microcontroller technology and applications 374

Appendices

A	Binary, hexadecimal and BCD	379
B	16F84 instruction set	386
C	A versatile microcontroller-based digital panel meter	388
D	Addressing modes, address decoding and the memory map	397
E	Answers to selected exercises	404

Index **407**

CHAPTER

1

Introducing embedded systems and the microcontroller

IN THIS CHAPTER

We are living in a second age of industrial revolution, when the availability and processing of information are causing untold changes in all our lives. While mankind has dreamed for many years of the possibility of building computing machines, the dream started to become a reality in the late 1940s. It was then that the first electronic computers, based on massive racks of thermionic valves, started their laborious calculations. In 1948 the transistor was invented, and the first integrated circuit was built in 1959. This set the stage for a spectacular process of electronic miniaturisation. Integrated circuits became more and more compact, enabling more and more circuitry to be placed on them.

In 1971 Intel produced the first microprocessor, the 4004, which handled data as 4-bit numbers, and contained 2250 transistors. It followed this soon with the 8008, and within a few years a number of companies were making their own microprocessor offerings. The age of the microprocessor had arrived! Very early in their development, and certainly by the end of the 1970s, two trends were emerging for these remarkable devices. One was to scale down, in size if not computing power, the general-purpose computer; this led quickly to the first desktop machines. The other, much more revolutionary, was to place the microprocessor in products which apparently had nothing to do with computing. They began to find their way into photocopiers, grocery scales, washing machines, and a host of other products, wherever there was a requirement to exercise some control function. While the first trend led to an inexorable demand for faster and bigger processors with increasingly sophisticated mathematical capability, the second placed lower demands on computational power and speed. It wanted physically small and cheap devices, with as much functionality of the system as possible squeezed onto one integrated circuit.

Such microprocessors became known as microcontrollers, and the systems they controlled, embedded systems.

Though humbler by far than their high-powered cousins, microcontrollers sell in far greater volume, and their impact has been enormous. To the electronic and system designer they offer huge opportunities.

This chapter starts our exploration of the fascinating and hidden world of embedded systems. We will meet first the embedded system itself, and discover something of its nature and characteristics. Then we will start our study of the intelligence inside the embedded system: the microprocessor or microcontroller.

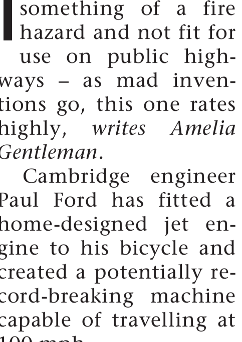
Specifically, the chapter aims:

- to introduce the embedded system and describe its characteristics
- to review prerequisite microprocessor knowledge, thereby defining a starting point
- to consider certain fundamental choices in microprocessor design
- to introduce the features of a general-purpose microcontroller
- to introduce three microcontroller families, which will be used as examples in parts of the book

1.1 Embedded systems and their characteristics

1.1.1 The essence of the embedded system

The newspaper article of Fig. 1.1 describes a jet-propelled bicycle that appeared on TV and in the newspapers. Apart from its extraordinary novelty value, the design overcame some very difficult technical problems. One reason for the success of the project was the inbuilt microcontroller system which kept the engine under control. Yet in a substantial write-up, the reporter makes no mention of this at all. Why doesn't the headline shout 'Microcontroller Tames new Jet Engine', or similar? The answer is simple. The reporter almost certainly didn't even know the microcontroller was there. Her attention was drawn entirely by the novel combination of a jet engine and a bike. The microcontroller was not visible, there was no indication of the presence of a computer, and she had no reason to believe there might be one involved. Nevertheless, the engine could not have functioned for more than a few seconds without the continuous action of the control system, which not only enabled successful operation, but also provided the condition monitoring to eliminate situations of danger. With a miniature jet engine there are plenty of those!



Inventor fears noise and heat will dull appeal of jet cycle

Paul Ford on his jet powered bike PHOTOGRAPH: TONY JEDREJ

Intolerably noisy, something of a fire hazard and not fit for use on public highways – as mad inventions go, this one rates highly, *writes Amelia Gentleman*.

Cambridge engineer Paul Ford has fitted a home-designed jet engine to his bicycle and created a potentially record-breaking machine capable of travelling at 100 mph.

Aside from its speed, the vehicle does have a couple of advantages: there is no need to pedal and jet paraffin is affordably priced.

But even the inventor accepts that these attractions are outweighed by the problems the prototype bike poses.

Primarily there is the noise. It emits 102 decibels when stationary and when it gets going it sounds like an aeroplane on take-off.

Then there is the heat. "You have to be careful not to stand behind the bike because the exhaust emerges at about 480 degrees centigrade – hot enough to burn the hairs off your body," Mr Ford warned.

And this is not a bicycle to delight the environmentalists. "We weren't trying to be particularly green. The emissions are what you would expect from a jet engine."

Mr Ford, aged 37, co-owner of a model aircraft shop in Cambridge, invented the miniature gas turbine engine. Although it can turn out 22 lbs of thrust – about equivalent to a moped engine – this generates high speeds because the bicycle itself is so light.

During preliminary tests at a disused airfield, the vehicle reached 55 mph at half power and Mr Ford is confident that, with a bit of work, 100 mph will be reached easily.

"I've been too scared to go any faster. At the moment the steering is extremely sensitive, something else that needs refining. To begin with I was also concerned that it might actually take off but the design seems to have prevented that risk."

While happy to accept that his invention is not practical, Mr Ford remains uncomfortable with the mad professor status the creation has forced on him.

"I'm pretty certain that this is the first jet powered bike in Britain. People thought it couldn't be done and I wanted to prove them wrong."

He added: "A lot of people have told me that it's a crazy thing to try to do, but I don't think it's eccentric at all" he said.

Figure 1.1 A high-speed embedded system: the jet-propelled bicycle. (From *The Guardian*, 12 May 1998. Reproduced by permission of *The Guardian* newspaper and the *Cambridge Evening News*.)

We call this type of control *embedded control* – and the overall system an *embedded system*. A definition of an embedded system is as follows:

> *An embedded system is a system whose principal function is not computational, but which is controlled by a computer embedded within it.*

The computer is likely to be a microprocessor or microcontroller. The word *embedded* implies that it lies inside the overall system, hidden from view, forming an integral part of a greater whole. One consequence of this is that the user may be unaware of the computer's existence. Another is that the computer is usually purpose designed, or at least customised, for the single function of controlling its system. If removed from the system it would be an odd assortment of printed circuit boards and/or integrated circuits, recognisable only to the specialist as something which might be called a computer.

Applying this definition tells us that a personal computer, even though it contains a microprocessor, is *not* an embedded system. Its end function is to compute. Even if the same computer was connected to a set of instruments, which it then controlled, that would not be an embedded system. If, however, the same computer was built permanently into an identifiable system, and customised so that its sole purpose was to control the one system (which may mean losing such apparently essential features as its case, keyboard, screen, or disk drives), then it would form part of an embedded system.

Embedded systems come in many forms and guises. They are extremely common in the home, the motor vehicle and the workplace. Most modern domestic appliances – washing machines, dishwashers, ovens, central heating and burglar alarms – are embedded systems. The motor car is full of them, in engine management, security (for example locking and anti-theft devices), air-conditioning, brakes, radio, and so on. They are found across industry and commerce, in machine control, factory automation, robotics, electronic commerce and office equipment. The list has almost no end, and it continues to grow.

Figure 1.2 re-expresses the embedded system definition as a simple block diagram. There is a set of inputs from the controlled system. Based on information supplied from these inputs, the controller computes certain outputs, which are connected to actuators within the system. There *may* be interaction with a user, e.g. via keypad and display, and there *may* be interaction with other sub-systems elsewhere, though neither of these is essential to the general concept.

In the jet-propelled bicycle the control system measures three variables from the engine (temperature, pressure and rotational speed), and also receives a control input from the driver. Its only output controls the fuel flow to the engine. From the inputs it first of all determines whether the engine is operating safely. If a danger condition is detected (for example the motor is running too hot or too fast), the controller takes emergency action. In the absence of a danger condition, it computes the appropriate drive signal for the fuel flow.

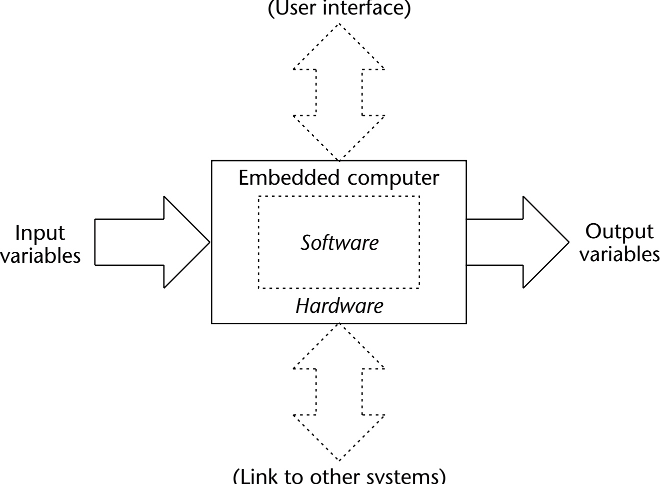


Figure 1.2 The essence of the embedded system.

1.1.2 Further features of the embedded system

1.1.2.1 Constituents of the embedded computer: hardware and software

As with all computer systems, the embedded computer is made up of hardware and software, as symbolised in Fig. 1.2. In the early days of microprocessors much of the design time was spent on the hardware, in defining address decoding, memory map, input/output and so on. When the hardware design was completed, a comparatively simple program was developed, limited in size and complexity by restricted program memory size, and the development tools available. Since then there have been huge strides in hardware development. Much of the hardware system is now contained on a single chip, in the form of a microcontroller, and developments in memory technology allow the use of much longer and more sophisticated programs. Hardware design of the computing core of the embedded system is now in many cases viewed as a comparatively straightforward affair. The design attention has shifted to some extent towards software development, with advanced languages and tools available to develop sophisticated programs.

1.1.2.2 Timeliness

The example jet engine is able to change its speed extremely fast, and can easily self-destruct. The controller must be able to respond fast enough to keep its operation within a safe region. This is a characteristic of operating in 'real time'; the controller must be able to respond to inputs as they happen and make responses within the time-frame set by the controlled system. This style of operation is different from the mode of operation, for

example, of a personal computer. While it may be annoying, you can tolerate waiting for your computer to refresh the graphics display or complete a computation. You cannot tolerate waiting while your car's anti-skid braking system decides whether or not to apply the brakes!

Some embedded systems operate within absolutely rigid time demands; for others the demands are less stringent. They all, however, exhibit the characteristics of timeliness: a need for the designer to understand fully the time demands of the controlled system and be responsive to them.

1.1.2.3 System interconnection

Figure 1.2 raises the possibility of interaction with other systems. While some embedded systems clearly need only one controller, others are likely to use several or many, each to control one sub-system. Necessary shared information is then passed between them by a simple network, devised to suit the needs of the overall system. A good example of this is the modern motor car. Though each of the 'embedded sub-systems' in it may be controlled by one microcontroller, they can all be linked together to form one overall interconnected system. This approach is made more attractive due to the extremely low cost of most commercial microcontrollers. A network of low-cost microcontrollers is often cheaper, and simpler to develop, than a single complex computer undertaking many tasks.

With the advent of the Internet, a generation of Internet-compatible embedded systems is emerging. The cooker, television and washing machine may soon be communicating together! It is anticipated that within a few years even the most simple of devices may be Internet-linked. The truly standalone device will then exist in a dwindling minority.

1.1.2.4 Reliability

Suppliers of software packages designed to run on Personal Computers release them on the market knowing that they are likely to contain software errors (bugs). It is vitally important to get them to market early, and fixes can always be distributed after the faults have been discovered. Suppliers of most embedded systems cannot afford this luxury. One significant software error in a car model could destroy the reputation of the manufacturer for ever. Therefore the embedded system designer must develop a good grasp of reliability issues, and how a reliable system can be achieved. This implies good design procedures in both hardware and software, coupled with systematic testing and commissioning.

1.1.2.5 The market-place

The market that the embedded system sells into is very competitive. As with other 'hi-tech' markets, the challenge is increased greatly by the very rapid advances of technology. New products may quickly be rendered obsolete by technological change, and thus potentially have very short life cycles. This lays the stress on excellent design and development strategy.

Adding all these features together, a second definition of the embedded system now follows, more descriptive and verbose. The technical features mentioned in this effectively lay down the agenda for this book.

An embedded system is a microcontroller-based, software-driven, reliable, real-time control system, autonomous, or human or network interactive, operating on diverse physical variables and in diverse environments, and sold into a competitive and cost-conscious market.

1.1.3 The skills of the embedded system designer

It is becoming clear that embedded systems have enormous variety, and call upon many technical disciplines. This is indeed one of the attractions of working with them. This multi-disciplinary nature is illustrated in Fig. 1.3.

A full understanding of the microcontrollers we will work with only comes with some knowledge of computer architecture and integrated circuit design and manufacture. The need for control, which inevitably implies measurement and actuation, leads us into further branches of electrical and electronic engineering. Associated with the measurement, we find a need for analogue as well as digital electronics. One could go on adding further disciplines, for example Digital Signal Processing or Electromagnetic Compatibility, to the diagram. These are also important to the embedded system, but will not claim much space in an introductory book like this.

1.2 Our starting point: the microprocessor

1.2.1 The microprocessor reviewed

Our approaching study of the microcontroller will rely on the reader having a reasonable knowledge of microprocessors. We will pause briefly to review this knowledge, to ensure a defined starting point. Figures 1.4–1.6

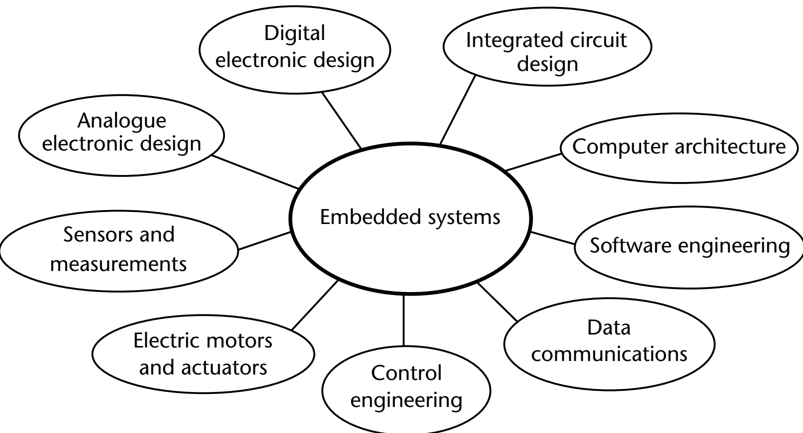


Figure 1.3 Embedded system design calls on many disciplines.

summarise what we need to know. The contents of each diagram will be briefly reviewed, but readers who need greater detail on these matters should consult an appropriate text, for example Refs. 1.1 and 1.2. Both of these have excellent introductory chapters on microprocessors. Appendix A contains a summary of binary arithmetic and counting schemes, which may also be worth reading at this stage.

A microprocessor is a simple computer, contained more or less in one integrated circuit (IC, also colloquially called a 'chip'). Like any computer it follows a sequence of instructions, known as a program. Each instruction causes a very simple action to take place, generally either a computation, a transfer of data or a decision. The microprocessor can perform each instruction extremely fast, so that by building on these very simple actions much more complex tasks can be undertaken.

A diagram of the hardware of a simple microprocessor-based system is shown in Fig. 1.4. The essential features are:

- the microprocessor
- a section of memory to store the program
- another section to store temporary data
- some contact with the outside world (through the input/output port)
- a means of interconnecting these elements (i.e. data and address bus, together with some control lines)

Program memory is usually stored in a form of memory called *ROM – Read-Only Memory*. Data memory is usually stored in a type of memory called *RAM – Random Access Memory*. ROM retains its contents when the system is powered down; RAM does not. Memories are defined according to size, generally in terms of numbers of bytes. For this the prefixes K- and M- (or Mega) have gained ubiquitous customary usage. These differ from the

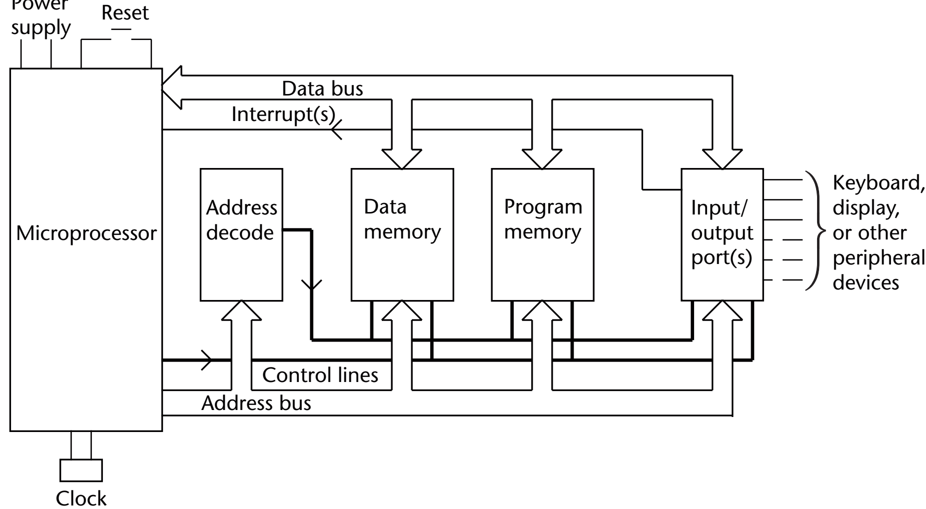


Figure 1.4 A simple microprocessor system.

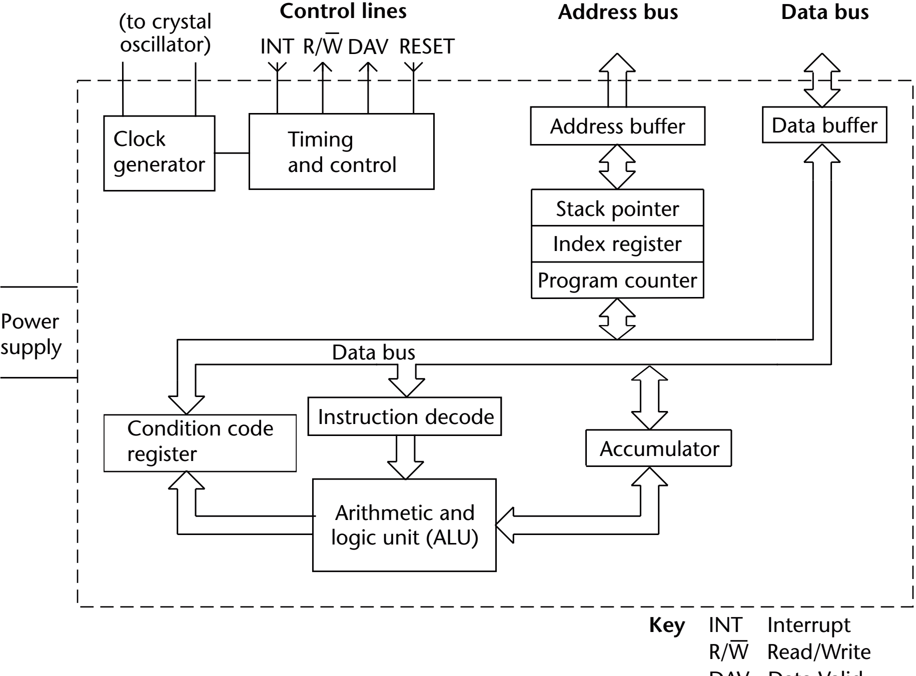


Figure 1.5 A typical microprocessor.

conventional decimal multipliers (e.g. the kilo- of kilometre or kilogram). K- indicates a multiplier of 2^{10}, i.e. 1024, while Mega is actually 1 048 576, i.e. 2^{20}. A memory of 4 Kbytes contains 4096 byte-sized locations.

A block diagram of a 'typical' imaginary microprocessor appears in Fig. 1.5. The computing function takes place in the *Arithmetic Logic Unit* (*ALU*), where arithmetic and logical operations take place. Part of the ALU is the *accumulator*. This is the register where the operand, the number on which the operation is being performed, is held. The size of the Accumulator, in number of bits, determines the size of number that the processor can operate on. It is reflected across the whole microcomputer system, for example in the size of the data bus and memory locations. The ALU, together with the control section around it, is known as the *Central Processing Unit* (*CPU*).

The action of the microprocessor is synchronised to the clock generator, often based on a quartz crystal oscillator. Any microprocessor can only operate within a certain range of clock frequencies, whose limits are set by the fabrication technology of the device and specified by the manufacturer. Each has a maximum (for microcontrollers usually in the range 4 MHz to around 30 MHz). Those based on *dynamic logic* (see Ref. 1.3 for further details) have a minimum as well. Those which can operate down to DC are known as 'fully static'.

The clock oscillator frequency is divided down within the microprocessor (generally by a factor between 4 and 12, depending on the microprocessor),

giving a lower internal operating frequency. One period of this internal frequency is sometimes called a *machine cycle*, or an *instruction cycle*. All instruction execution is made up of integer numbers of machine or instruction cycles.

In normal system operation the processor works down the list of instructions which make up the program. It fetches each one from program memory, decodes it with its Instruction Decode circuit, and then executes it. The instruction is in many cases accompanied by further pieces of code, also stored in program memory, which are treated as operand data, or addresses where the operand data may be found.

The microprocessor 'keeps its place' in the program by means of the *Program Counter*, which always holds the address of the next instruction to be executed. In order to fetch the next instruction, the processor places the value held in the Program Counter on the address bus, and signals through the control lines that it wishes to read data. Memory corresponding to that address will, upon receiving the address and control signals, place the instruction word on the data bus, which the processor can then read. As each word is read from program memory, the Program Counter is incremented.

Figure 1.6 illustrates this sequence of activities, for the processor of Fig. 1.5 and for a certain instruction, as a timing diagram. It can be seen that there are four clock cycles in each machine cycle. The first cycle shown is an 'instruction fetch' cycle. The address of the instruction to be fetched is placed on the address bus, and the $R/\overline{W}$ line indicates that the data transfer is to be a 'read'. In response the addressed memory places data onto the bus. This is received by the microprocessor and decoded by the Instruction

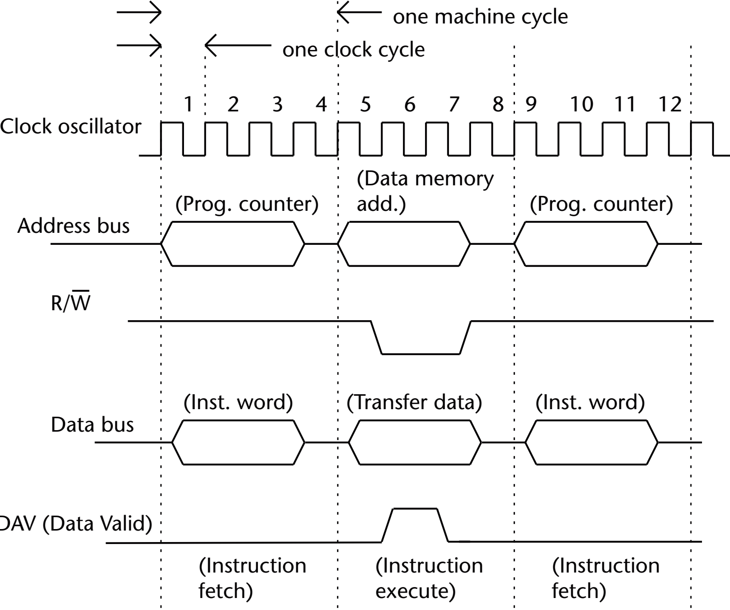


Figure 1.6 The microprocessor fetch/execute cycle.

Decode circuit. In the second machine cycle the instruction is executed; the example illustrates a data move from processor to memory. The processor sets values on the address and data buses, and signals a write by setting the $R/\overline{W}$ line low. The DAV line goes high to indicate that the bus data is valid. The falling edge of this signal is used to latch the data into memory. This particular instruction has taken two machine cycles to complete. It is then followed by the Instruction Fetch cycle of the next instruction. It follows that simple microprocessor operation can be seen as a relentless cycle of instruction fetch, decode and execute.

1.2.2 More on instructions, and the ALU

A typical 8-bit ALU is able to perform the operations shown in Table 1.1. Using combinations of these very simple operations, almost any other mathematical function can be implemented, albeit sometimes laboriously.

Each processor (or processor family) has its own instruction set, from which the program is written. Each instruction is a binary word, known individually as the *op code* (operation code), or collectively as *machine code*. The processor CPU can recognise and respond to these codes. The instruction set is the collection of all these op codes. It uses the basic ALU operations listed earlier, and adds to these certain data transfer and branch instructions. This gives an instruction set the following typical instruction categories:

Table 1.1 What an ALU can do.

Increment A	A = A plus 1
Decrement A	A = A – 1
Add A to M	A = A plus M
Subtract M from A	A = A – M
*AND A with M	$A = A \cdot M$
*OR A with M	A = A + M
*Exclusive OR A with M	$A = A \oplus M$
Shift A left	A = 2A
Shift A right	A = A/2
Rotate A left	
Rotate A right	
Complement A	NOT A
Clear A	A = 0

A represents the contents of the accumulator; M is a number held in memory.
The statement 'A = ' implies 'A becomes' (original value of the accumulator overwritten).
*The logical function is performed between corresponding bits of the two operands.

- *Data transfer*: instructions which move data from one register or memory location to another.
- *Arithmetic*: instructions which perform arithmetic operations between specified data words.
- *Logical*: instructions which perform logical functions between specified data bits or words, for example INVERT, AND, OR, Rotate.
- *Program branch*: instructions which cause a program to deviate from simple sequential execution of instructions held in program memory, for example as a subroutine call or return, or conditional branch.[1]

The result of an operation undertaken in an accumulator frequently exceeds the range of the number which can be held in the Accumulator. Therefore associated with the ALU is a 'Flag Register'; this contains a number of bits which give further information about the result of the previous instruction. It is known as the *Status Register* (Microchip Inc.), *Condition Code Register* (Motorola), or *Programme Status Word* (Intel and Philips). These bits may include:

- a **zero** bit, indicating whether the result was zero
- a **carry** bit, indicating whether there was a carry from the most significant bit (msb) of the accumulator, also used as a 'borrow' in subtraction
- a **sign**, or **negative** bit, indicating whether the result was negative (interpreting the result in *two's complement* arithmetic[2]) – hence this bit is simply set to the msb of the result
- a **half-carry** bit, indicating whether there was a carry between the lower and higher nibbles of the result – this is useful for *Binary Coded Decimal* (BCD) arithmetic
- an **overflow** bit, indicating whether the two's complement range has been exceeded. It is set if there has been a carry out of bit 7 but not bit 6, *or* a carry out of bit 6 but not bit 7
- a **parity** flag, indicating whether an odd or even number of 1 bits are in the accumulator

As there are not usually enough 'condition code' flags to fill an 8-bit register, many processors use the remaining few bits for other purposes, for example interrupt mask bits or register bank address bits. These will be considered later.

---

1 A conditional branch instruction tests a certain condition of the microprocessor system, for example a register bit. It transfers program operation to a different program section if the test condition is met, and continues the program in sequence if it is not.

2 'Two's complement' is a means of expressing negative numbers in binary. Read Appendix A if you are unfamiliar with it.

WORKED EXAMPLE 1.1

The hexadecimal numbers F8 (= 11111000 in binary) and 09 (= 00001001 in binary) are added in an ALU. Determine the result and what Condition Code Flags are set (assume the ALU has all the above flags except the Parity flag). Interpret the result in unsigned binary and in two's complement.

Solution: Figure 1.7 illustrates the addition, together with the flag settings. The 8-bit unsigned binary result is valid only if the state of the carry flag (C) is noted. The result is valid in two's complement (hence OV is zero), and is a non-zero (Z = 0) positive number, as indicated by the state of the N flag. A half-carry (HC) has also occurred.

									Unsigned binary	Two's complement
	1	1	1	1	1	0	0	0	F8	–8
+	0	0	0	0	1	0	0	1	09	+9
=	0	0	0	0	0	0	0	1	01	+1

C HC

C	HC	Z	N	OV
1	1	0	0	0

Condition code flags

Figure 1.7 Example addition of two binary numbers.

1.3 Some microprocessor design options

We go on to consider some aspects of microprocessor design which go beyond the basic structure assumed so far. These aspects are discussed at a level appropriate to the small-scale microprocessor or controller; their application in larger computers is far more sophisticated. Readers who wish to gain further background in these areas are referred to Refs. 1.4 and 1.5.

1.3.1 Von Neumann and Harvard

In the conventional von Neumann architecture, program and data memory share the same address and data buses, and are hence both within the same memory map. This is illustrated in very simple form in Fig. 1.8(a). This approach is simple, robust and practical, and has been widely and successfully applied. If data memory is being accessed, program memory lies idle,

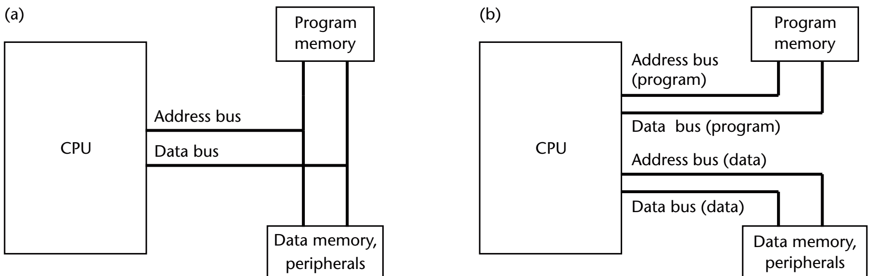


Figure 1.8 (a) The conventional von Neumann structure; (b) the Harvard structure.

and vice versa. Once the overall memory space is defined, it is up to the user to decide which area is allocated to data, and which to program. The structure does however lead to the 'von Neumann bottleneck'; time-sharing the data bus between both instruction and data means that maximum speed of executing a program will always be limited, as each has to use the bus in turn.

It is, however, possible to have more than one address and data bus, and hence to place data and program memory in different memory maps. This approach, sometimes called a Harvard structure, is shown in simple form in Fig 1.8(b). Instructions can now be fetched independently from, and if necessary simultaneously with instruction execution, thereby eliminating the von Neumann bottleneck. The two data buses can now be of different sizes, as can the two address buses. This allows each to be optimised for its own use, and has important implications in certain processor structures. The structure facilitates pipelining (see below), and also enhances program security. It is less likely that an errant processor will attempt to overwrite its own program, or jump into data memory and start interpreting data as instructions.

With its multiplicity of buses, this architecture does lead to a more complex hardware realisation than conventional von Neumann. Moreover, not every memory use is clearly divided into 'data' or 'program'. Look-up tables (i.e. tables of constant data, defined within the program), for example, may be embedded in program memory, but required for use as data.

1.3.2 Instruction sets – CISC and RISC

In simple terms, the operations that the designer of the microprocessor has at her initial disposal are those listed in Table 1.1. The microprocessor instruction set *could* be based on these, and thus they would be available to the programmer in their 'raw' form. Alternatively, it is possible to group them together in simple combinations, so that an instruction from the instruction set is actually interpreted by the CPU as a sequence of perhaps two or three of these primitive instructions. This practice is known as *microcoding*, and the task of interpreting each program instruction into instruction primitives is done by a ROM internal to the CPU.

Many early microprocessor designers adopted this practice, and tried to create instructions for every possible eventuality. This appeared to approach a sophisticated and 'ideal' machine. A processor of this type gained the name Complex Instruction Set Computer (CISC). One of its more obvious characteristics is that code for different instructions can be of quite different lengths, and have widely differing execution times. The CISC processor also occupies more space on the IC, due to the requirement for internal ROM.

Studies of CISC instruction usage revealed, however, that in a 'typical' program most of the instructions were not being used for most of the time (e.g. 80% of programs were made up of 20% of the instruction set). It was therefore reasoned that if the most-used instructions were optimised in terms of speed, and the others removed (but with their function still achievable by combinations of those that remained), then program execution time could be reduced and the CPU design simplified. In parallel with this, technological advances, especially in the area of high-density memory, meant that the pressure to minimise program code length was no longer so great.

The result was a 'back to basics' move, leading to the simpler but faster Reduced Instruction Set Computer (RISC). This has the following characteristics:

1. *The CPU does not make use of microcoding*.
2. *Memory is accessed via load and store instructions only*; the requirement to operate on memory contents is achieved by multiple instructions.
3. *All instructions are executed in one machine cycle*; this means that each instruction must be represented by one word only – hence all op codes must be equal to or within the instruction bus size, and must include the operand within them.

RISC machines have the advantages of simplicity and speed, but carry the apparent disadvantage that their program code is almost invariably longer and more complex. With memory becoming ever cheaper and of higher density, and with more efficient compilers for program code generation, this disadvantage is diminishing.

1.3.3 Instruction pipelining

As Fig. 1.6 showed, conventional microprocessor program execution is a relentless sequential cycle of instruction fetch, decode and execute. For a given processor the only way of speeding up this operation is by speeding up the clock.

Consider an alternative: that as one instruction is being executed, the next is already being fetched. If this is done, the instruction throughput can be dramatically increased without reducing the actual instruction execute time. This is the basis of pipelining – it's a simple idea which can make processors run much faster, but it does place certain strict requirements on the nature of the instructions.

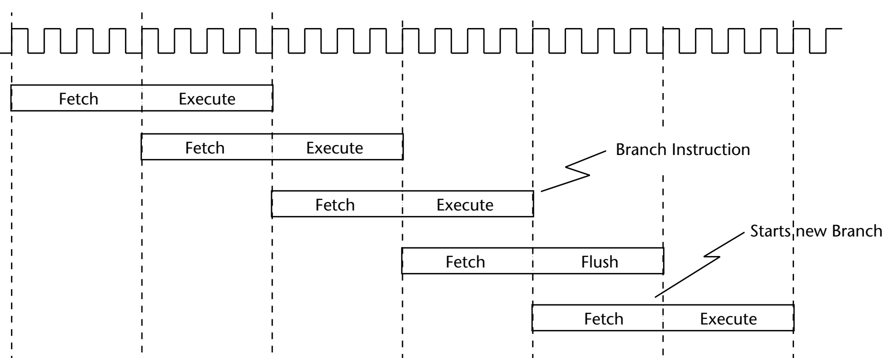


Figure 1.9 Pipelined instruction execution.

In order to work, *all* the instructions of the processor must have the same duration of execution, and it must be possible to split the fetch–decode–execute cycle for *all* individual instructions into a number of stages of equal duration. Then, as any one instruction enters its second stage, the following instruction enters its first. This is illustrated in Fig. 1.9, for instructions divided into just two stages (i.e. fetch and execute). As one instruction is executing, the next is already being fetched. It can be seen that the instruction throughput for the first three instructions is twice as fast as in Fig. 1.6. If the instructions had been broken into three stages, it would have been three times as fast, and so on.

Simple pipelining fails at conditional program branches. When the processor is executing a branch instruction, it is already fetching the next instruction in the program, but if the branch does take place that next instruction is no longer needed. So it must 'flush out' that instruction, and fetch the one where the branch starts. This is why branch instructions often take longer in a pipelined architecture. The example of Fig. 1.9 shows two instructions being successfully fetched and executed. The third is a branch, and the fourth instruction, though fetched, is never executed, and one machine cycle is lost. The fifth instruction shown is from the start of the program section to where the branch has taken place.

1.4 The microcontroller: its applications and environment

A microcontroller is a particular type of microprocessor, optimised to perform control functions for the lowest cost and at the smallest size possible. Generally microcontrollers are used in a recognisably 'embedded system' environment.

There is a huge range of microcontroller applications. Some are drawn from volume markets – the motor car, domestic appliances, mobile phones and toys. These applications are sold in such high volume that dedicated controllers are frequently developed for them. Others, like medical or scientific instruments, are sold in smaller numbers, and are more likely to make use of the wide variety of general-purpose controllers that are

available. At one extreme of complexity, simple (and very cheap) controllers are used to replace 'glue logic' in a digital system. At the other extreme, advanced 32-bit controllers perform sophisticated signal processing activities.

1.4.1 Microcontroller characteristics

Arising from their 'embedded control' environment, microcontrollers usually have the following features:

- input/output intensive, i.e. they are capable of direct interface to a significant number of sensors and actuators
- a high level of integration, with many peripheral[3] devices included 'on-chip'
- physically small
- comparatively simple program and data storage requirements
- ability to operate in the real-time environment
- an instruction set optimised for the embedded environment, e.g. yielding compact code, limited arithmetic and addressing capability, strong in bit manipulation
- low cost

In many microcontroller applications either or both of the following features are also essential:

- an ability to operate in hostile environments, for example of high or low temperature, or high electromagnetic radiation;
- a low power capability, and features which ease the use of battery power.

In today's fast-moving world, both the manufacturers of microcontrollers and the people who design with them work under a complex set of sometimes conflicting forces. On the one hand, semiconductor technology is advancing inexorably. Every year it becomes possible to integrate more onto a single IC, and to do this more cheaply, with the chip operating at faster speed and lower power. Interacting with this technological change are powerful market forces, spurring on the development by demanding new capabilities from the microcontrollers. Against this, however, is set a certain conservative tendency. Companies using the microcontrollers have invested time and money in supporting work with a particular device, and don't want all this wasted when they move on to its more powerful successor.

---

3 A point of terminology: the term 'peripheral' in the computer world used to (and still does) refer to equipment which worked peripheral to the computer, such as printers and modems. In the microprocessor world it was adopted to describe such off-chip devices as I/O ports and serial links. Now these peripherals have become integrated onto the controller chip itself, but we still call them peripherals.

The outcome of all this is that the manufacturer usually develops a family of microcontrollers all based around one core, where the core contains the CPU and its surrounding control features (i.e. essentially the features of the early microprocessor, as in Fig. 1.5). The core defines the instruction set, and hence keeping the core design constant ensures software compatibility between different members of a processor family. To the core, and on the same IC, can be added the peripheral devices which seem best to meet a particular need. Even though the microprocessor world is one of such great change, many microcontrollers can trace their history very directly back for over 20 years! Once a company has committed itself to designing with a particular microcontroller family, it is reluctant to change, but looks to the manufacturer to supply it with the necessary technological advances, based around a familiar core. Infrequently, the manufacturer makes a step change by introducing a new core.

1.4.2 Features of a general-purpose microcontroller

Every microcontroller is different, and each has its own unique combination of core and peripherals. Figure 1.10 shows, in block diagram form and with no interconnections, the features which might be found in a simple general-purpose controller. The core is the element that remains constant for the whole family built around it. Ideally all memory is on-chip, and several different memory technologies may be applied to meet the differing needs of program and data storage. Interconnection to the outside world is through a number of parallel and serial ports. A counter/timer is available for event counting, or to measure or generate timing intervals.

1.4.3 Some example controllers

There are a huge number of different microcontrollers now on the market, and it is not easy to select a small number of representative devices to illustrate the hardware principles described in this text. Three example microcontroller families have been chosen, and in keeping with the introductory nature of the book they are all 8-bit devices. The families selected are well established in industry, and are chosen to illustrate the variety of approaches taken to solve common problems. Each one carries at least one

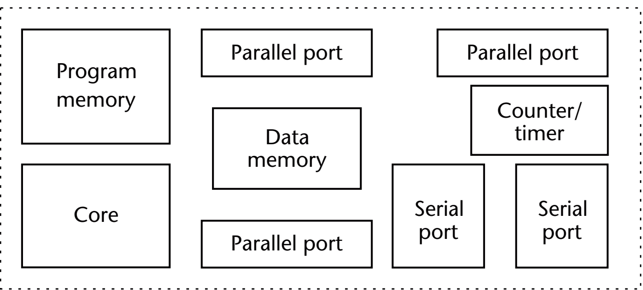


Figure 1.10 An example microcontroller block diagram.

feature not shared with the others. The example controllers are not, however, selected or treated as equals. First of all, the PIC 16F84 device is very small and low-cost, and used for some of the simplest possible embedded systems. The 80C552, on the other hand, is a comparatively complex (and more costly) device, rich in peripherals and with extensive memory addressing capability. The 68HC05 and 68HC08 lie in complexity between these two. Second, to encourage in-depth knowledge to be built up of one microcontroller, most early examples are of the PIC 16F84. Later examples are drawn more generally from the microcontrollers named, as well as one or two other close relatives.

It should be understood that the selection of these devices as examples is not intended to compare them in a competitive way (in the sense of seeking out ‘the best’), nor does it necessarily represent an endorsement of any one of them.

It is recommended that the reader obtains at least the full data sheet of the 16F84 (Ref. 1.6). It will also be useful, but not essential, to have access to data on one or more of the others (Refs. 1.7 and 1.8). There are also many very useful Application Notes, published by the manufacturers, as well as books targeted towards individual devices, for example Refs. 1.9 and 1.10.

The remainder of this chapter introduces the three example families, looking particularly at their background, architecture and CPU. In the coming chapters we build on this introduction by looking at certain features of these microcontrollers in greater detail.

1.5 Microchip Inc. and the PIC™ microcontroller

1.5.1 Background, and meet the family

It was the General Instruments Corporation, back in the late 1970s, that first produced the PIC microcontroller (Ref. 1.11). In its early years it did not make a wide impact. The design was later taken over by Microchip Inc., and PICs are now one of the fastest moving families in the 8-bit arena, in more senses than one. First, they run very fast; second, the family is growing at a tremendous rate; and third, at the time of writing Microchip only operates with 8-bit controllers, and therefore has a special interest in making this controller size as attractive as possible. PICs cover a very wide range of 8-bit operation. At the lower end, they are simpler, cheaper and smaller than most devices that the competition can offer, and are thus used in situations where controllers would not be thought of as the right solution, even down to simple glue logic applications. At the high end, however, they are quite ready to take on the best of the 8-bit competition, with sophisticated devices equipped with excellent peripherals. PICs have made themselves particularly attractive to the student and low-budget developer. Development tools (both hardware and software) are cheap and readily available, and Microchip is very supportive of the novice designer.

Table 1.2 PIC microcontroller families.

Family	Program word size	Number of instructions	Minimum instruction execution time
12CXXX	12/14-bit	33/35	400 ns
16C5X	12-bit	33	200 ns
16C/FXXX	14-bit	35	200 ns
17CXXX	16-bit	58	120 ns
18CXXX	16-bit (enhanced)	77	100 ns

Microchip offers five closely related families of microcontroller, as shown in Table 1.2. PIC examples in this book are taken mostly from the 16F84, with a limited number from the more sophisticated 16C74.

All PIC controllers use a RISC-like structure, with Harvard architecture and pipelined instruction execution. This leads to one of the strengths of the PIC family: a very high instruction throughput.

1.5.2 A 16F84 microcontroller overview

The block diagram of this controller is shown in Fig. 1.11(a), and the IC pin connections in Fig. 1.11(b). The structure is radically different from other microcontroller families that we will see. The program memory area can be seen at the top left of the diagram. Its 13-bit address input is derived from the Program Counter, and its 14-bit data output forms the instruction word (transferred on the *Program Bus*). Note that although an address bus of this size can address 8K words, the actual program memory size is 1K. To the right of this is the RAM area, made up of 68 8-bit locations. Microchip calls these memory locations *file registers*. This memory has its own 7-bit address bus (again, the potential range of this bus is not fully exploited). Its data input/output is linked to the 8-bit microcontroller data bus. With the address and data buses to program memory and RAM being separate and independent (and of different sizes), we can conclude that this is a Harvard organisation. To add to the addressing complication, the EEPROM memory area has its own address source held in a dedicated RAM register (called EEADR). Data is transferred through another register EEDATA. The stack forms yet another distinct area of memory, with only eight 13-bit locations. It has no connection to the data bus, and cannot be used for temporary data storage.

As with all other 16XXX microcontrollers, the 16F84 makes no attempt to allow conventional memory expansion. Data and address buses are simply not 'bonded out' to external pins, and there is no chance to extend internal memory within the microcontroller memory map.

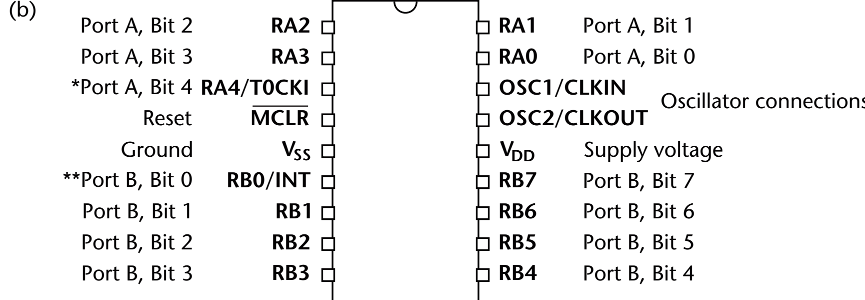
As peripherals, the 16F84 has:

- two parallel ports, one of 5 bits (Port A) and the other of 8 (Port B)
- one 8-bit Counter/Timer (TMR0)

(a)

Flash/ROM Program Memory PIC16F83/CR83 512 x 14 PIC16F84/CR84 1K x 14
13
Program Counter
Data Bus
8
EEPROM Data Memory
8 Level Stack (13-bit)
RAM File Registers PIC16F83/CR83 36 x 8 PIC16F84/CR84 68 x 8
EEDATA
EEPROM Data Memory 64 x 8
Program Bus
14
7
RAM Addr
EEADR
Instruction reg
Addr Mux
5
Direct Addr
7
Indirect Addr
TMR0
FSR reg
RA4/T0CKI
STATUS reg
8
MUX
8
I/O Ports
Instruction Decode & Control
Power-up Timer
Oscillator Start-up Timer
Power-on Reset
Watchdog Timer
ALU
RA3:RA0
RB7:RB1
Timing Generation
W reg
RB0/INT
OSC2/CLKOUT
OSC1/CLKIN
MCLR
VDD, VSS
MUX: Multiplexer

(b)



*also Counter/Timer clock input
**also external interrupt input

Figure 1.11 The PIC 16F8X. (a) Block diagram; (b) pin connections: plastic dual-inline package (PDIP). Part (a) reprinted with permission of the copyright owner, Microchip Technology Incorporated © 2001. All rights reserved.

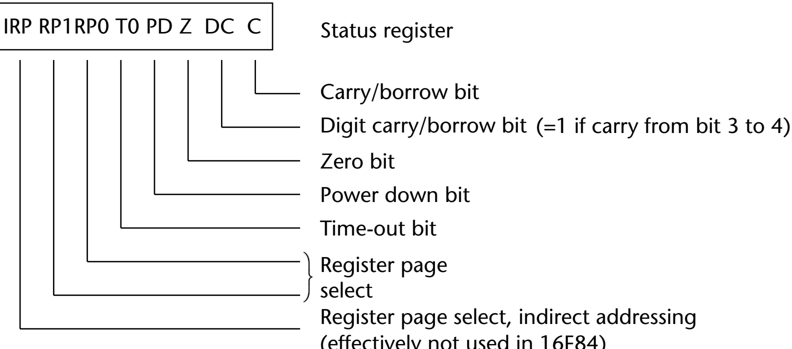


Figure 1.12 The 16F84 Status Register.

1.5.3 CPU, programming model and instruction set

Manufacturers of complex microprocessors usually supply a *programming model*. This is a simplified diagram of those internal microprocessor registers that are of direct interest to the programmer. Owing to its comparative simplicity, a programming model of the 16F84 is not normally necessary; the block diagram of Fig 1.11(a) is used in its entirety.

The ALU can be seen in the lower part of Fig 1.11(a). There is a single accumulator (the Working, or W, register), and most operations are performed between it and either the contents of a RAM file register, or with operand data embedded within the op code. The Status Register (Fig. 1.12) holds three bits (Z, DC and C) which give status information about the result of the most recent instruction executed.

The 16F84 has 35 instructions in its instruction set, a summary of which appears in Appendix B. These will be explored in detail in Chapter 3. It can operate at any clock frequency up to a maximum of 10 MHz. Each instruction cycle is made up of four oscillator cycles. The resulting fastest instruction execution time is therefore 400 ns.

1.6 The Philips 80C552 microcontroller

1.6.1 Background, and meet the family

As Intel was the first company to produce a microprocessor, it seems right that it was also the first to produce a microcontroller. It did this in 1976 with the MCS-48 (appearing in three versions, the 8035, 8048 and 8748; Ref. 1.11). In its time the MCS-48 was revolutionary. The 8748 had on-chip ultraviolet erasable programmable read-only memory (EPROM), 64 bytes of RAM, and three input/output ports. It attracted many adherents. In 1980 Intel launched its successor, the 8051. The 8051 took up where the ’48 left off, and has also become firmly embedded, in more senses than one, in the microcontroller world. Though the 8051 itself is now an old device, many companies (for example Atmel, Dallas, Philips and Siemens) offer

controllers based on the '51 core, and further developments are repeatedly being produced.

As one of the manufacturers who have adopted the 8051 design, Philips has developed many variants. The 80C51 is a CMOS version of the 8051, and Philips has extended this into a wide-ranging family. Its practice has been to use the *whole* of the 80C51 as core, and to add further peripherals to this.

1.6.2 Controller overview and architecture

The 80C552 is an advanced member of the Philips 80C51 family. It has a powerful collection of on-chip peripherals, and is one of three versions of the 8XC552, which differ only according to the program memory available on-chip. The main features are:

Within the 80C51 'core'

- Four 8-bit parallel ports
- Two 16-bit Counter/Timers
- One UART (Universal Asynchronous Receiver Transmitter)

Extra to 80C51 Core

- Two 8-bit parallel ports
- One 16-bit Counter/Timer with Capture and Compare enhancements
- Two PWM (Pulse Width Modulation) ports
- One I^2C (Inter-Integrated Circuit) serial port
- 256 bytes of static RAM (data memory)
- Eight-input 10-bit ADC (Analogue to Digital Converter)
- 8 Kbytes mask programmable ROM, 256 bytes RAM (83C552)
  or 8 Kbyte EPROM, 256 bytes RAM (87C552)
  or as 83C552, without ROM (80C552)

The 8XC552 block diagram is shown in Fig. 1.13. The 80C51 core can be seen to the left, and all the peripherals listed above can easily be distinguished. All sub-systems within the controller are served by a single data bus. By sacrificing two of the parallel ports, the data and address buses *can* be made available to the outside world. In this case, the lower 8 bits of the address bus are multiplexed with the data bus. This is essential for the 80C552, which has no internal program memory, but usually unnecessary for the 83C552 and the 87C552, which are self-contained.

The '552 is offered in three speed ranges, with clock frequencies of up to 16 MHz, 24 MHz and 30 MHz. In each case the minimum clock frequency is 1.2 MHz. The clock is divided internally by 12 to form one machine cycle.

1.6.3 CPU, programming model and instruction set

A programming model of all members of the 80C51 family is shown in Fig. 1.14. All the registers shown (excluding the Program Counter) are memory-

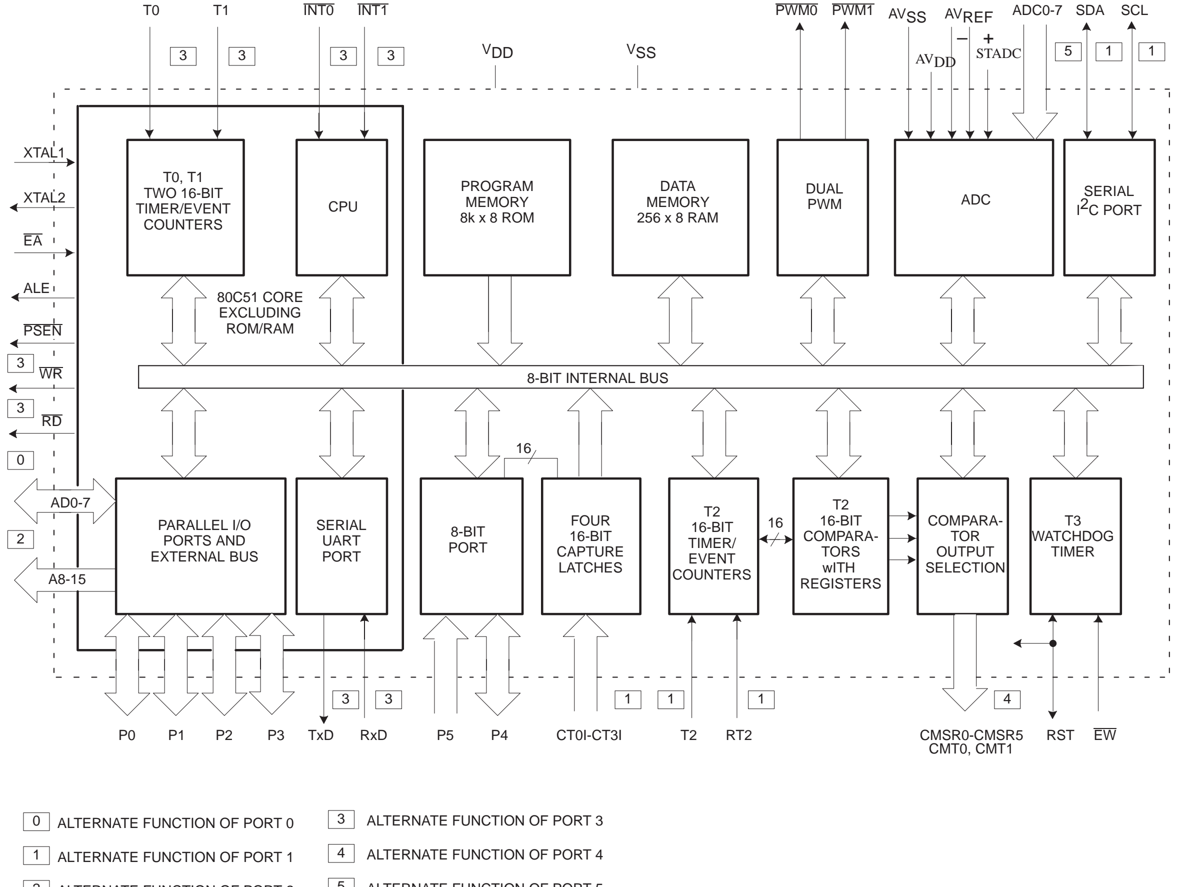


Figure 1.13 The 80C552 block diagram. (Reproduced by permission of Philips Semiconductors.)

mapped. There is a single 8-bit accumulator. There is also a B register, used only during 8-bit multiply and divide instructions. The Stack Pointer is 8-bit, and may be used to locate the stack theoretically anywhere in the 256 bytes of on-chip RAM. Available stack space is effectively restricted by other uses to which the programmer wishes to put the RAM. The Program Status Word (PSW, equivalent to the Status Register of the 16F84) has 4 bits (highlighted in the figure) which indicate arithmetic status.

An interesting feature of the 80C51 CPU is that it includes a 1-bit Boolean processor, which uses the PSW Carry bit as the accumulator. This allows a complete set of instructions on single-bit operands, including Boolean operations, as well as move, set and clear. These operations can be used on a block of bits in the static RAM, as well as certain locations in the peripheral controls (including all parallel port bits).

All 80C51 variants use the same (CISC) instruction set, which contains 50 distinct assembler mnemonics. When adapted to the different addressing modes available, these amount to 111 instructions. These are listed in the

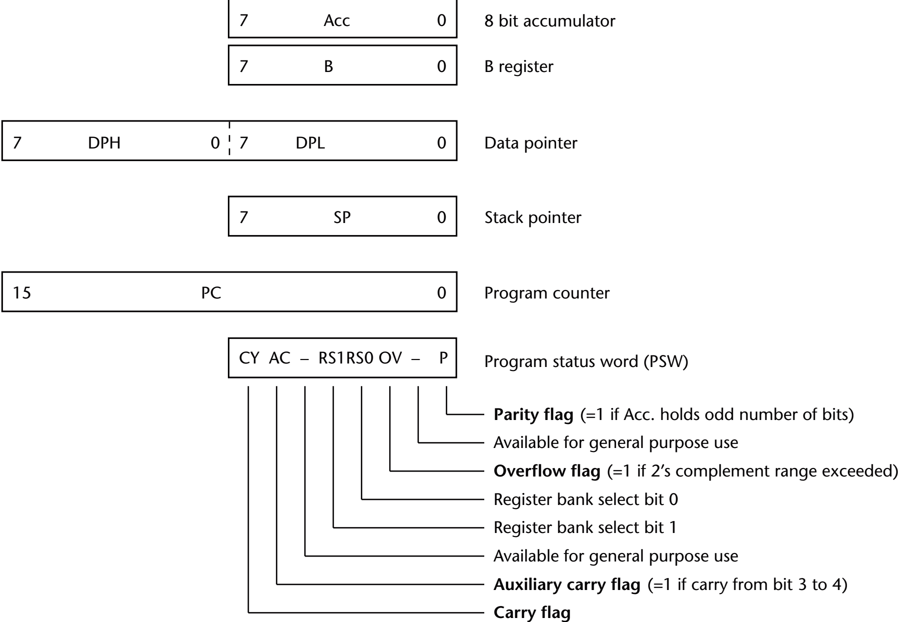


Figure 1.14 The 80C51 programming model.

categories: arithmetic (including 8-bit by 8-bit hardware multiply and divide), logical, data transfer, Boolean variable manipulation, and program branching. Instructions are encoded in one, two or three bytes, and execute in from one to four machine cycles (i.e. 12 to 48 oscillator cycles).

1.7 The Motorola 68HC05/08 microcontrollers

1.7.1 Background, and meet the family

Motorola was early in the microprocessor field, but was not the first. By the time it entered, with the 6800, it was able to offer a device which enjoyed remarkable longevity. From the 6800 it developed further 8-bit conventional microprocessors (e.g. the 6809), and also a number of single-chip controllers, starting with the 6801. These led to the 68HC11, a sophisticated and widely used microcontroller. The HC infix indicates the new high-speed CMOS (Complementary Metal Oxide Semiconductor) technology with which it is made. From the 68HC11 the 68HC12 and 68HC16, both 16-bit controllers, have been developed.

An indirect development of the 6800 family was the 6805 (M146805 in full), available initially in HMOS (High-Density N-Channel MOS) and CMOS versions. Here the CPU was simplified, for example by the removal of the second (B) accumulator of the 6800, reduction in addressing capability, and consequent reduction of certain register sizes. As one of the earlier CMOS controllers, the 6805 had a great impact on low-power

applications. The 6805 was subsequently upgrade and reissued using ‘HC’ CMOS technology. This has enjoyed very widespread use as a simple and low-cost microcontroller. Motorola claims that over 2 billion (2×10^9) units of the 68HC05 have been sold. The number of variants are too many to list, but contain devices targeted specifically for automotive, computer, consumer, industrial, telecommunications, TV and video applications.

Since the late 1990s the 68HC05 has been in the process of being replaced by the 68HC08, which provides a direct upgrade. Both the ’05 and the ’08 use the ‘7’ infix to indicate EPROM or OTP (One-Time Programmable) memory version (e.g. the 68HC705P) and the ‘9’ infix to indicate Flash memory.

All of the 68HCXX microcontroller families have some similarity in architecture and instruction sets, so it is a comparatively easy task to move from one to another, selecting the device most appropriate for the job.

1.7.2 68HC05 controller overview

The block diagram of the general purpose ‘B’ version of the 68HC05 is shown in Fig. 1.15. The main features are:

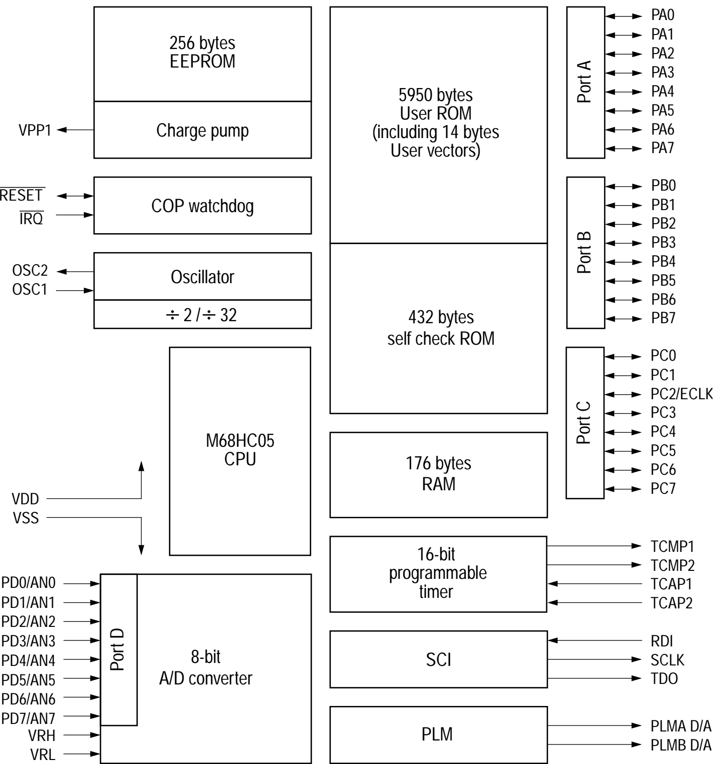


Figure 1.15 The 68HC05B6 block diagram. (Copyright of Motorola Inc. Used by permission.)

- three 8-bit parallel input/output ports and one 8-bit input-only port
- one 16-bit timer system
- one Serial Communications Interface (SCI)
- eight-channel 8-bit Analog to Digital Converter
- 'Computer Operating Properly' (COP) Watchdog System
- two Pulse Length Modulation (PLM) outputs, intended for digital to analogue conversion
- 256 bytes of EEPROM
- 176 bytes of RAM
- 5950 bytes of User ROM

The bus structure, though not shown, follows a simple von Neumann pattern. Like the 16F84 there is no external bus connection. There is a single memory map, within which *all* memory and addressable registers lie. Although it is not normal operation, programs *can* be executed from RAM or EEPROM, and there is a mechanism for loading either of these memory areas through the serial port.

The 68HC05 can operate with an internal clock frequency up to 2.1 MHz. This is divided down by 2 from the external clock. A SLOW mode of operation is also available, in which the user may insert a further divide-by-16 in the clock generator. This is useful for low-power applications. With a fully static design, it can operate down to DC.

1.7.3 CPU, programming model and instruction set

The 68HC05 programming model is shown in Fig. 1.16. There is a single Accumulator, an 8-bit Index Register and a 16-bit Program Counter. The Stack Pointer is initially set to $00FF_{16}$, and counts down as data is entered onto the stack. As the 10 most significant bits are fixed as shown, the actual range of the stack is from $00FF_{16}$ down to $00C0_{16}$, giving 64_{10} actual memory locations.

The CPU has 62 basic instructions. These are the same as the earlier M146805, with the addition of an unsigned hardware multiply, in which the contents of the Accumulator and Index Register are multiplied together. A wide range of addressing modes are supported, which when applied to the basic instructions gives a final total of 210. All instructions are encoded in one, two or three bytes, and execute in from two to eleven internal clock cycles.

With four bits in the Condition Code Register which indicate arithmetic status, program branching can take place on any of the conditions $<$, $\leq$, $=$, $\neq$, $\geq$, $>$, negative and positive. Bit set and clear, and bit test and branch instructions are supported for operands in the first page of memory (i.e. the first 256 bytes). Boolean operations between bits are not available.

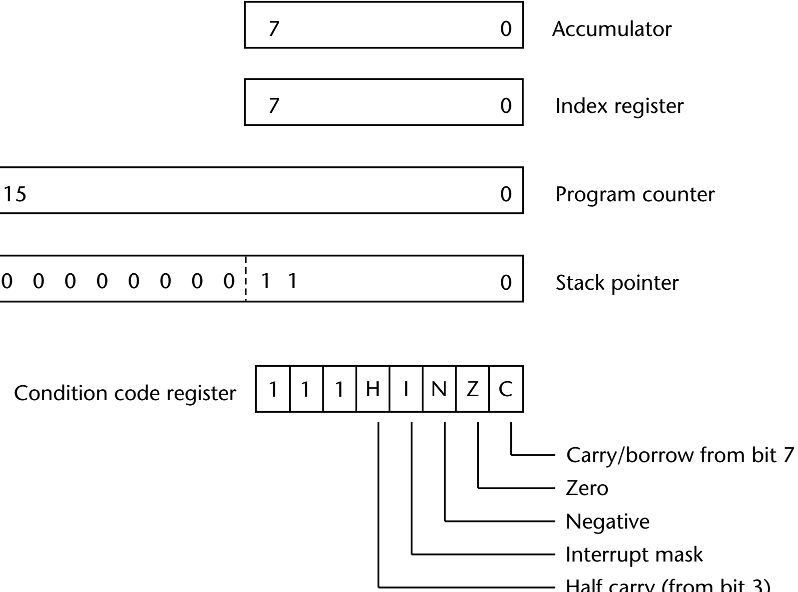


Figure 1.16 The 68HC05 programming model.

1.7.4 The 68HC08

The 68HC08 family of microcontrollers offers a direct upgrade to the 68HC05. The CPU has been expanded, some new instructions have been added (for example a 16-bit by 8-bit divide), and higher operating speeds are possible. The most significant advance that the '08 offers, however, is its inclusion of Flash memory, which allows greatly increased programming flexibility. Like the '05, the '08 is available in many versions, targeted towards specific areas of application. These are identified by an alphabetical suffix to the device number, which include **-GP** (general-purpose), **-AS** (automotive), and **-JL/JK** (low-cost general-purpose).

The programming model of the '08 is shown in Fig. 1.17. Index and Stack registers are now 16 bit, and the Condition Code Register is enhanced by the addition of a two's complement overflow flag.

SUMMARY

1. An embedded system incorporates a computing element, typically a microprocessor or microcontroller, to perform a control function. Many embedded systems are small and low-cost, and are aimed towards the volume market. They apply recognised hardware and software principles to meet the particular requirements of the embedded environment.

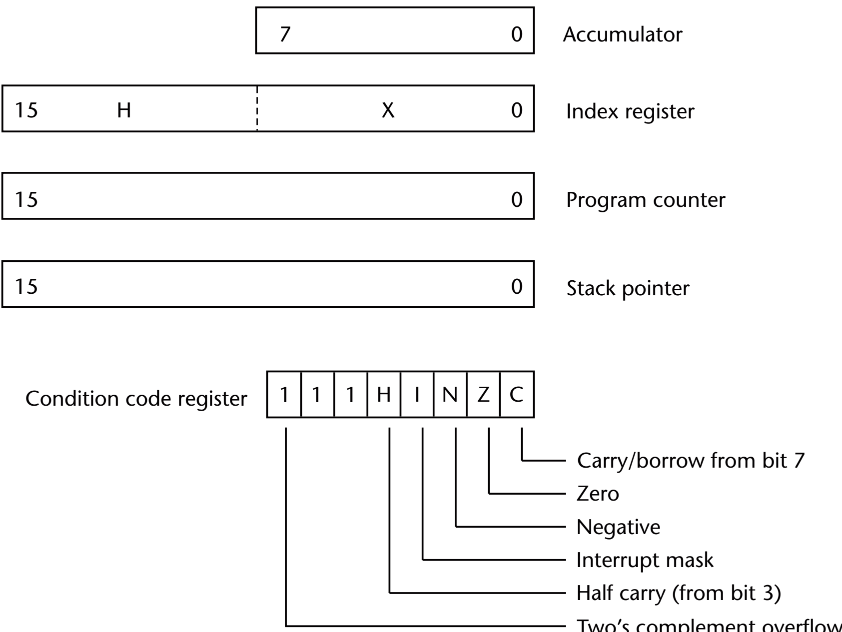


Figure 1.17 The 68HC08 programming model.

2. The microprocessor was one of the great technological revolutions of the 20th century. It is, however, based on principles that are now well established and stable. Owing to technological advances, faster and more powerful processors are continuously being introduced.
3. The microcontroller is a microprocessor intended for small-scale control applications. It integrates a conventional microprocessor core and a range of peripheral devices on a single IC, at the smallest size and lowest cost possible. A family of controllers is based around the same core, but with different peripherals and IC packaging, optimised for different applications.
4. While all microprocessors differ, there are some fundamentally different options in processor design, which have major significance for the final perfomance. These options include RISC vs. CISC, conventional von Neumann vs. Harvard, and the option of pipelining.
5. The PIC, 80C51 and MC68HC05/08 series of microcontrollers are all successful and well-established 8-bit controllers, each with their own unique attributes and advantages.

REFERENCES

1.1 Horowitz, P. and Hill, W. (1989) *The Art of Electronics*, 2nd edn. Cambridge: Cambridge University Press. Chapters 10 and 11.

1.2 Storey, N. (1989) *Electronics, a Systems Approach*, 2nd edn. Reading: Addison-Wesley. Chapter 12.

1.3 Sedra, A. and Smith, K (1998) *MicroElectronic Circuits*, 4th edn. Oxford: Oxford University Press.

1.4 Clements, A. (1991) *The Principles of Computer Hardware*. Oxford: Oxford Science Publications.

1.5 Hennessy, J. L. and Patterson, D. A. (1998) *Computer Organisation and Design*. San Mateo, CA: Morgan Kaufmann.

1.6 Data on Microchip PIC 16F84: `http://www.microchip.com/10/Lit/PICmicro/16f8x/index.htm`

1.7 Data on Philips 80C51 and family: `http://www.philips.semiconductors.com/mcu/products/`

1.8 Data on Motorola 68HC05/08: `http://www.mcu.motsps.com/hc08/index`

1.9 Peatman, J. B. (1997) *Design with PIC Microcontrollers*. Englewood Cliffs, NJ: Prentice Hall.

1.10 MacKenzie, S. (1999) *The 8051 Microcontroller*, 3rd edn. Englewood Cliffs, NJ: Prentice Hall.

1.11 Bursky, D. (ed.) (1978) *Microprocessor Data Manual*. Philadelphia, PA: Hayden.

EXERCISES

1.1 Find a non-technical friend and describe to him or her what an embedded system is and what its common characteristics are.

1.2 List five products or product sub-systems which could be embedded systems, choosing examples from the domestic, automotive, industrial or office environments. For each one, outline briefly the effects of unreliable performance.

1.3 For the products listed in Exercise 1.2, consider the importance of time in the operation of each. Is *fast* operation required? Does the system operate within strict time demands?

1.4 The Harvard memory structure gives some clear advantages over the conventional von Neumann structure. Can you think of any disadvantages? (Consider and expand on: system complexity, flexibility of memory utilisation, ease of accessing data tables in program memory, access to Stack.)

1.5 Three microprocessors, A, B and C, have maximum clock speeds respectively of 10 MHz, 24 MHz and 20 MHz. Processor A divides its clock by 4 to give one machine cycle, processor B by 12, and processor C by 8. A and B take two machine cycles to perform an 'add accumulator to immediate data' instruction, while C takes three cycles. Place the processors in order of the speed in which they can perform this instruction.

1.6 An 80C552 can execute an 'add register to accumulator' in one machine cycle. A PIC 16F84 can also perform this in one machine (instruction)

cycle. If the PIC is running at a clock frequency of 10 MHz, at what speed should the 80C552 run in order for them to execute the instruction in the same time?

1.7 For the 16F84, the 80C51, the 68HC05 and the 68HC08, draw up a table showing all arithmetic and logical flags appearing in their 'Status Registers' (or equivalent), indicating which flag is implemented in which microcontroller. Which register has the potential to give the most information?

1.8 The now obsolete Intersil IM6100 microprocessor had a 12 bit data bus and a 12 bit address bus.

(a) How many words of data could it address?
(b) Assuming 12 bits in every memory location, how many bits of data could it address?
(c) With the data coded in two's complement, what range of numbers could it represent?

(Refer to Appendix A if you are initially unable to do this question.)

1.9 How long do the longest and the shortest instructions take to execute, for each of the 16F84, 80C552, and 68HC05, when each is operating at
(a) its fastest clock frequency?
(b) an external clock frequency of 2 MHz?

1.10 In the 16F84 Instruction Set (Appendix B), identify all instructions which operate on single bits.
(a) Which bits may be used as operands?
(b) What operations on the operand bits are possible?
(c) Describe clearly how operand bits are identified. How many bits within the instruction are needed for this?

Index

Entries printed in italic are the names of companies.

ADC *see* Analogue to Digital Conversion
address decoding 399–403
addressing modes 397–9
Advanced Micro Devices
  27C128 119–20
aliasing 136–7
ALU *see* Arithmetic Logic Unit
amplifiers 138
Analog Devices
  OP42 389–90
Analogue to Digital Conversion 144 ff.
  linearity errors 146–7
  microcontroller ADCs 154–9
  quantisation error 145–6
  resolution 145–6
  selecting an ADC 160–1
  without ADC 153–4, 388 ff.
  see also Successive Approximation ADC
Arithmetic Logic Unit 11
assembler programming 66 ff.
  data types 89–92
  file structure 76–8
  format 74–6
  vs. high-level languages 206–8
  macros 84
  program layout 84–5
Atmel
  AT89C1051 313–15
  AT29C010A 123–4
averaging *see* sampled data

background debug mode 370–1
batteries 295–8
Batron
  BT10809 390–1
BCD *see* Binary Coded Decimal
BDM *see* background debug mode
binary
  addition and subtraction 380
  binary to BCD conversion 329–36
  fractional numbers 322–3
  number representation 379–80
  rounding 337–9
  truncation 337–9
  see also fixed point
bit banging *see* serial communication
Binary Coded Decimal 382
  BCD to binary conversion 329–36
brown out *see* power supply
buffer (data) 93–4
Burr Brown
  DAC714 389–91
  INA114 138, 388, 390
bus
  data and address 8
  multiplexed 114

C++ 209
CAN *see* Controller Area Network
CCR *see* Condition Code Register
charge pump *see* converter
CISC *see* Complex Instruction Set Computer
Clock oscillator 57–8
CMOS 310–13
commissioning
  of minimum system 94–7
  strategies 365–9
Complex Instruction Set Computer 14

Condition Code Register 12–13
construction (of circuit) 61
Controller Area Network bus 183–6
converter (voltage)
  boost 305–6
  buck 303–5
  charge pump 307
Counter 47–8
Counter/Timer
  auto reload 52–3
  Capture Register 237
  compare 237–8
  event counting 50
  frequency measurement 236, 332–5
  repetitive interrupts 51–2, 88–9
  timing 50–1, 233–5
C programming language 209 ff.
  in embedded environment 214–18
  MISRA C 218–19

DAC *see* Digital to Analogue Conversion
data acquisition 136 ff.
DC motor 278–82
delay *see* timing
design 349 ff.
  conceptual design 356–8
  detail design 365
  of embedded systems 352–4
  embodiment design 358–65
  of engineering products 350–2
  hardware layout 360–2
  hardware/software trade-offs 359–60
  specification development 353–6
  for test 364–5
differentiation *see* sampled data
Digital to Analogue Conversion 128 ff.
  serial DAC 129
  see also Pulse Width Modulation
displays 261 ff.
  LCD 263–8
  seven segment LED 262–4
DRAM *see* dynamic RAM
dynamic RAM 106–7

EIA-232 181–3
Electrically Erasable Programmable Read-Only Memory (EEPROM) 110–12, 121–3
embedded system
  definition 4,7
  essential features 2–7
Erasable Programmable Read-Only Memory (EPROM) 109–10, 119–20
  emulator 369–70

fault finding 96–7
filtering (analogue) 131, 139–41
  anti-aliasing 140–1
fixed point 322 ff.
  addition 324–6
  division 328–30, 334–5
  fractional numbers 322–3
  multiplication 326–8, 333–5
  overflow (of range) 325–6
  subtraction 324–6
  see also binary
Flash (memory) 112–13, 123–4
floating point 335–7
flow diagrams 78–9
  module flow diagram 202–3
frequency measurement *see* Counter/Timer

Harvard architecture 13–14
H Bridge *see* switching
hexadecimal 380–1
High-Level Languages (HLLs) 206–9
Hitachi
  HD44780 217,265–8
  HM6264 121
host computer 66

I^2C bus 177–81
  port 187–8
ICE *see* In-Circuit Emulator
In-Circuit Emulator 371–2
Indexed addressing 90,398
inductive loads, switching of 273–5
  see also switching
Integrated Development Environment (IDE) 74, 215
integration (numerical) *see* sampled data
interface, digital 253–6
  see also signal conditioning
Inter-Integrated Circuit *see* I^2C bus
interrupt
  context saving 232–3
  critical regions 232–3
  on change 40
  latency 227–32
  masking 232–3
  prioritisation 226–7
  review 34,36
  service routines (ISRs) 83–4
  vector 67–9,83
  see also under timing
ISO Open System Interconnect 174–5

Jackson's Structured Design (JSD) 203–6
Java 209

latency *see* Interrupt
LCD *see* displays
Light-Emitting Diode (LED) 45–6
Linear Technology
LTC1062 141
Liquid Crystal Display *see* displays
logic analyser 372–4
low power (design for) 314–17

Maxim
MAX232 183
MAX274 141
MAX538/MAX539 129, 176
MAX639 304–5
MAX690A 309
memory 102 ff.
array 105–6
ICs 118–24
implementation 113–15
maps 67–8, 399–403
program 115–16
review 103–4
see also under IC manufacturer's names
Microchip Technology Inc. 19–20
16C72, PWM generation 133–5
16C74, ADC 154–7
16F84
ADC, as part of 388 ff.
addressing modes 70–3
computed `goto` 80
configuration word 57
as display driver 262–4
EEPROM addressing 72
instruction set 68–74, 386–7
INTCON register 37
interrupt structure 37–8, 225, 226, 228, 229–30
memory programming 116–19
oscillator 61
overview 20–2
ports 39–42
power consumption 313–14
program memory map 68
special function registers 35
Status Register 21–2
table read 91–2
TIMER0 module 48–9
Watchdog Timer 240
PIC, families 19–20
microcontroller
choosing 362–4
general features 16–18
technological trends 374–5
microprocessor (review) 7–13
Microwire 175–7
MISRA C *see* C
monitor 370
motor (DC) *see* DC motor
Motorola
68HC05 25–8
addressing modes 397–9
memory map 67, 69
68HC08 28–9
68HC11 25
ADC 154–5
interrupts 225
power consumption 313
Watchdog Timer 240
MPASM 74–5
MPLAB™-CXX 215–17
multiplexer (analogue) 139
multi-tasking 241 ff.
simple programming solutions 242–4
see also RTOS

National Semiconductor
COP800 175
LM35 163
LP2951 301–2
Nyquist (Sampling Theorem) 137

offset binary 382–3
one time programmable 110
op amps 138
opto-isolator 253, 269
opto-sensor
reflective 270
slotted 270
oscillator 57–61
ceramic 60–1
quartz 59–60
R–C 58–9
OTP *see* one time programmable

PIC *see under Microchip Technology Inc.*
Philips Semiconductors
8051/80C51 22–5
80C552 22–5
ADC 154–60
I^2C port 187–8, 267–8
interrupts 225
power consumption 313–14
PWM generation 133–4
Timer 2 238–9
Watchdog Timer 240
PCF8574 179–80, 266–8
pipelining 15–16
ports (parallel) 38–9
quasi-bidirectional 42–3, 46–7
power supply 54–6, 293 ff.
brown out 308
supervision 307–9

see also batteries
prediction *see* sampled data
Program Status Word 12–13, 24–5
Pulse Width Modulation
current control in inductive load 277–9
hardware generation of 133–5
low pass filtering of 131, 135
principle of 130–1
software generation of 132–3
PWM *see* Pulse Width Modulation

quartz crystal *see* oscillator
quasi-bidirectional *see* ports
queues (data) 93–4

Random Access Memory (RAM) 8, 104
Read-Only Memory 8, 104
real time 224–5
real-time operating systems 245–9
scheduling 246–7
tasks 247
Reduced Instruction Set Computer 14
reference (voltage) 147–8
regulator (voltage) 298 ff.
linear 299–302
switching 302–6
relay 275–6
reset 54–6
resolution
of DAC 128–9
of ADC 145–6
RISC *see* Reduced Instruction Set Computer
ROM *see* Read-Only Memory
ROM emulator *see* EPROM
rounding *see* binary
RS-232 *see* EIA-232
RTOS *see* real-time operating systems

sampled data
averaging 338–41
differentiation 341–4
integration 345–6
prediction 344
sample and hold 142–4
serial communication 166 ff.
asynchronous 168
ports 188–90
bit banging 190–2
overview 167–9
physical limitations 169–73
protocols 173–5
synchronous 167
ports 186–8
serial peripheral interface 175–7
SGS
L297/L298 277, 289
shaft encoder 271–3
signal conditioning 137
simulator (instruction set) 94–6
software
development aims 198–9
development procedures 199–200
life cycle 197–8
modules 202
program flow 200–2
top-down decomposition 203
version control 219
special function registers 33–4
SPI *see* serial peripheral interface
SRAM *see* static RAM
stack 92–3
static RAM 107–8, 121
Status Register 12–13
see also 16F84 *under Microchip Technology Inc.*
stepping (stepper) motor 282 ff.
drive waveforms 285–6
drive system 288–90
phase switching 287
principles 282–5
speed profiles 286–7
see also inductive loads
strings (data) 90–1
subroutines 82–3
successive approximation ADC 149–51, 388 ff.
switched capacitor 151–3
see also Analogue to Digital Conversion
switches 43–4
arrays 257
debouncing 253, 254–6
see also switching
switching
reversible (H Bridge) 276–7
transistors as switches 259–61
see also inductive loads

tables (data) 90–2
target system 66
test *see* commissioning
timing
hardware-generated delays 87–8
software-generated delays 86–7
repetitive interrupts 51–2, 88–9
trouble-shooting *see* fault finding
truncation *see* binary
two's complement 382–5

Von Neumann 13–14
  bottleneck 114

Watchdog Timer (WDT) 53–4, 240–1, 308

weak pull-up 41

Xicor
  X24165 121–3