

Appendix A

Software Planning Using Flowcharts

A.1 Introduction

In this section we present an overview of flowcharts. Flowcharts are one of the oldest and most popular tools used to express an algorithm, regardless of whether it is a software or a hardware algorithm. Probably the reason flowcharts are so popular is because they give a general picture of the algorithm, as opposed to the sometimes obscure picture provided by other tools. Although, as mentioned before, flowcharts are used to express an algorithm that could later be implemented in either software or hardware, this appendix was written with the idea of a software implementation in mind.

Flowcharts are not intended to include every detail of the implementation of the algorithm because that is left to the actual coding of the instructions in the particular programming language of choice. Thus, flowcharts are expected to give a general idea of the algorithm and it will help to keep this in mind when developing algorithms using flowcharts. On the other hand, if you already know the programming language in which the algorithm will be implemented, then you could take advantage of it and introduce some details particular to that programming language, but bear in mind that these details might make it harder for the algorithm to be later implemented using some other programming language.

A.2 Symbols

We will be introducing five (5) basic symbols used to develop flowcharts. The names of these symbols are:

1. Start/Finish
2. Assignment or process
3. Subprocess

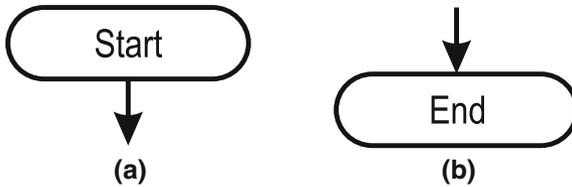
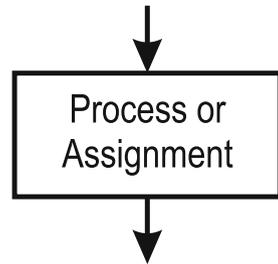


Fig. A.1 Start and Finish Symbols

Fig. A.2 Assignment or Process Symbol



4. Connector
5. Decision

The Start/Finish symbol consists of an oval shape as illustrated in Fig. A.1. As its name implies, the Start/Finish symbol is used to show an entry point into the algorithm or an exit point out of the algorithm. Although there is nothing that hinders from doing otherwise, it is a good programming practice to specify a single point of entry and also a single point of exit and we strongly recommend it. Otherwise a rather difficult to follow algorithm will result with an also difficult to maintain implementation, whether it is a software or a hardware implementation. When it is being used as an entry point it is labeled Start. Conversely, when it is used as an exit point it is labeled Finish, Exit, or End. If the symbol is used as an entry point into a subprocess or subroutine, then the name of the subprocess is written after the corresponding label, e.g. Start Sub1, Finish Sub1. A start/Finish symbol have only one arrow, either leaving a Start or entering a Finish, as shown in Fig. A.1.

Our next symbol is the Assignment or Process. This is drawn as a rectangle and inside of the rectangle we indicate the corresponding arithmetic or logic assignments including any expressions. We could include as many assignment statements inside the rectangle as we want, but if it begins to look crowded then it is a good practice to break it into several smaller rectangles which are easier on the eye. The assignment or process symbol is shown in Fig. A.2. Note that process symbols can have only one input arrows and one exit arrow.

The Subprocess symbol is shown in Fig. A.3. It looks a lot like the assignment symbol since it is also a rectangle, but it includes an additional vertical line on both the left and right side. This symbol is used to indicate whenever a subroutine or function call is invoked. After the subprocess returns, the algorithm continues with the next symbol in sequence in the flowchart. It is a good practice to use subprocesses in the algorithm development because it makes the algorithm less tedious and consequently

Fig. A.3 Subroutine or Subprocess Symbol

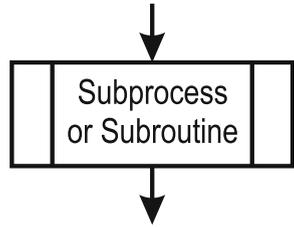


Fig. A.4 Connector Symbol

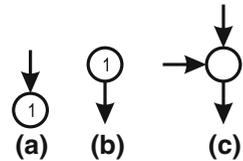
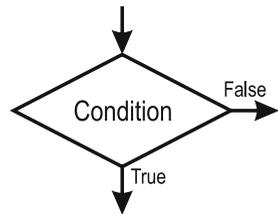


Fig. A.5 Decision Symbol



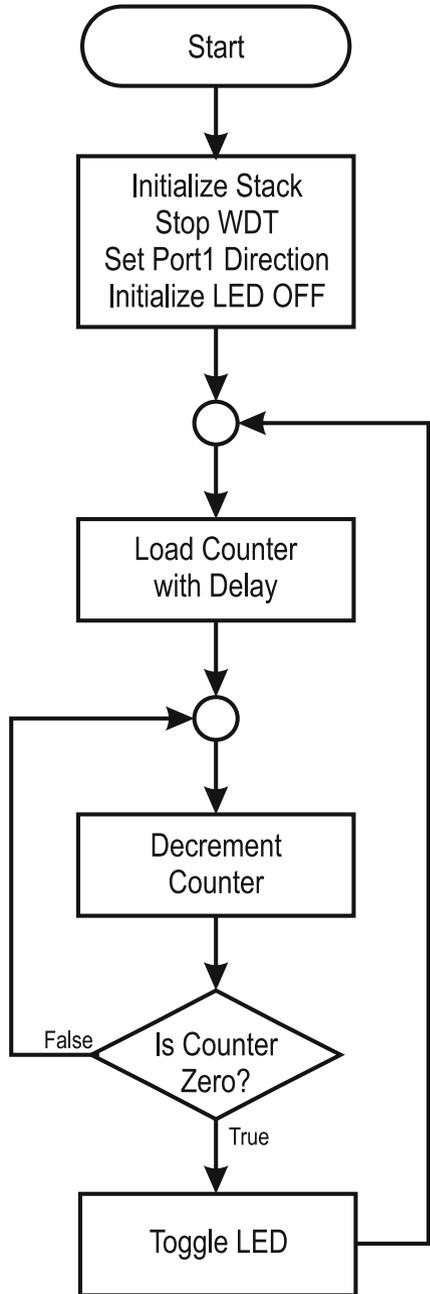
more appealing, especially if the subprocess is called upon several times. On the other hand, we should bear in mind that calling a subprocess and returning from it adds execution time to our algorithm. Like process symbols, subprocesses have only one input and one exit arrow.

The Connector symbol shown in Fig. A.4 is used whenever we run out of space in the page where we are developing our flowchart and need to continue on another part of the page or on a completely different page. Figure A.4a illustrates a connector leaving a page, while Fig. A.4b corresponds to the entry into a new page. Connectors are also used to denote junction points where multiple flows merge. This is the format illustrated in Fig. A.4c. Page connector symbols are used in pairs, both with the same identifier number or letter inside to indicate where the flow continues.

Our final but not less important symbol is the Decision symbol, see Fig. A.5. Its shape resembles a diamond and, as its name implies, it is used to alter the flow of the algorithm based on the value of some variable after a particular decision is made. The ability to make decisions allow for powerful algorithms as opposed to purely sequential ones. Decision blocks are the only ones that feature two outputs, one corresponding to each of the two possible decision outcomes: true or false, yes or no, left or right, etc.

Developing a flowchart for an algorithm involves these symbols and connecting them as needed with arrows indicating the flow direction.

Fig. A.6 Flowchart for Toggling LED



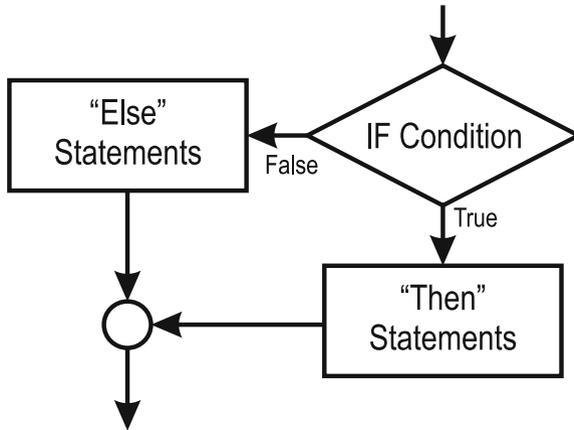


Fig. A.7 If-Else or If-Then-Else Structure

A.3 Putting it all Together

We will now develop an algorithm for toggling the red LED connected to pin P1.3 in the MSP430G2 Launchpad. The algorithm is illustrated in Fig. A.6 and a possible implementation using the MSP430 assembly language is shown in Listing A.1.

Listing A.1 MSP430 Assembly Language Instructions for Figure A.6

```

1 RESET  mov    #0280h,SP      ; Initialize stackpointer
2 StopWDT mov  #WDTPW+WDTHOLD, &WDTCTL ; Stop WDT
3 SetupP1 bis.b #1,&P1DIR     ; P1.0 as output
4 Main   xor.b #1,&P1OUT     ; Toggle P1.0
5 Wait   mov   #50000,R15    ; Delay to R15
6 L1     dec   R15           ; Decrement R15
7       jnz   L1            ; Delay over?
8       jmp   Main          ; Again
    
```

Observe that the algorithm in Fig. A.6 does not have an exit symbol. This characteristic is common in the main loop of many embedded programs: they run forever, as long as the system is powered. Exceptions include the case when the system is placed in a low-power mode. Chapter 7 illustrates several flowchart instances like this. Now that we have introduced the basic flowchart symbols and have shown a simple example we are only left with showing a few examples of how to combine the basic symbols to obtain common algorithmic structures.

An If-Else structure, is also known as an If-Then-Else can be obtained by combining the symbols for the Decision and Process as shown in Fig. A.7. This is a powerful construct and is used widely as part of the design of algorithms.

We could combine the same symbols and come up with one of the most used iterative structures, the while construct, shown in Fig.A.8 in its two variants: do-while and while-do.

We can of course combine symbols with structures as shown in Fig. A.9. What needs to be kept in mind is the fact that flowcharts are a tool that should be used to

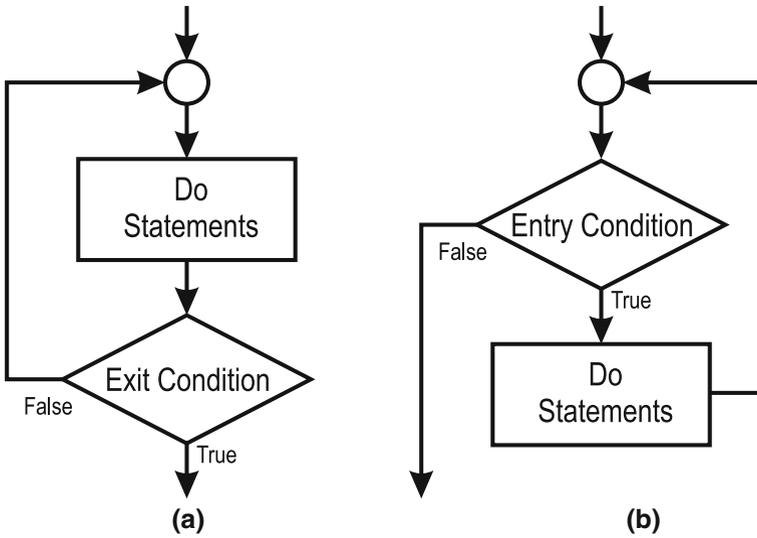


Fig. A.8 While structures: a. “Do-while”, b. “While-do”

aid in the design of algorithms to aid during the coding phase. As such we should strive to not make things more complex than they need to be, which includes both the flowchart and coding. The structure of the flowchart is such that it aims to have a single entry point and a single termination point. This includes not just the main program, but also any subroutine invoked by it or by any other subroutine. Avoid the so called spaghetti code which is both unreadable and unmaintainable. Structured techniques are the key for good algorithm development and programming practices. The power of these techniques are not to be underestimated.

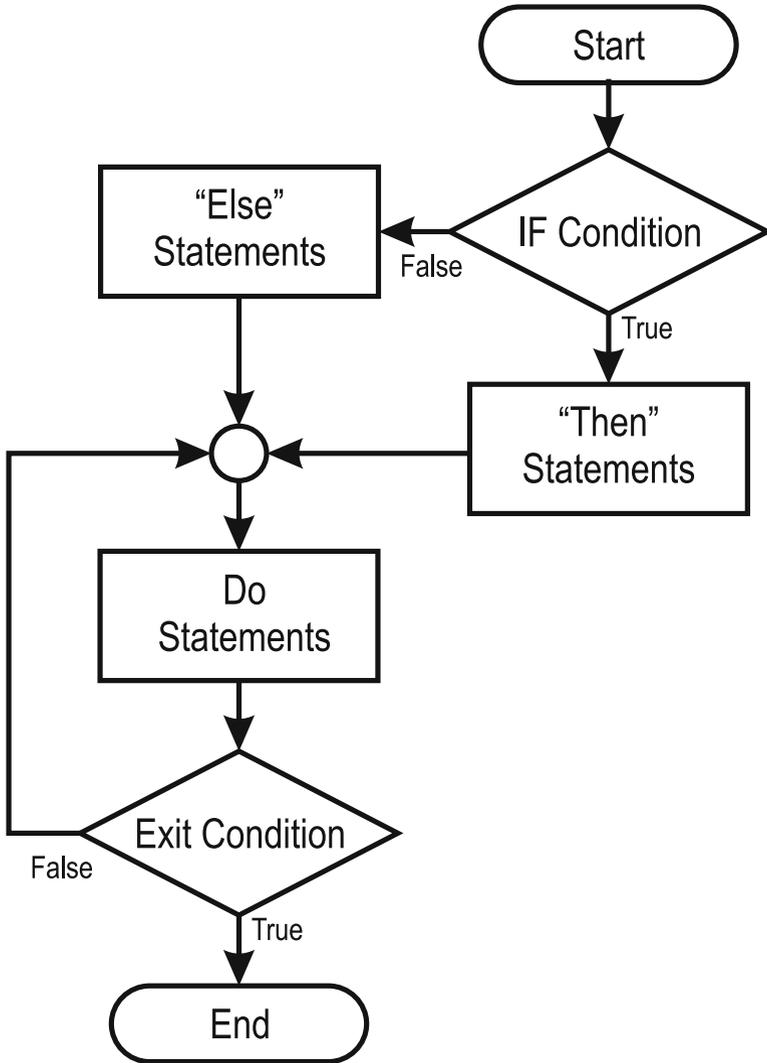


Fig. A.9 Combined Structure

Appendix B

MSP430 Instruction Set

B.1 Preliminary Notes

This appendix offers the complete instruction list for the MSP430 CPU and CPUX models. The contents of this appendix is courtesy of Texas Instruments Inc., and practically a reproduction of selected pages and chapters from MSP430x4xx Family (2007) and the MSP430x5xx Family (2008) user guides. Some additional paragraphs or sentences might be added for further clarifications, but otherwise the reproduction is verbatim. The appendix can be used as a quick reference for explanation and syntax of instructions, both for the normal and extended type.

Section B.2 describes the machine language instructions for the original MSP430 CPU. These may consist of one, two or three words, depending on the addressing modes and the type of instruction. Then the following section has the list of all the MSP430 instructions. Extended instructions are considered together with the CPUX assembly instructions in Appendix D.

The MSP430 instructions are the original 27 core instructions of the MSP430 CPU. For models with up to 64K memory range, these are the only ones available. For other models, these instructions may be used throughout the 1-MB memory range unless their 16-bit capability is exceeded. The MSP430X instructions are used when the addressing of the operands or the data length exceeds the 16-bit capability of the MSP430 instructions. There are three possibilities when choosing between an MSP430 and MSP430X instruction:

1. To use only the MSP430 instructions (except CALLA and the RETA instructions.) This can be done if a few, simple rules are met:
 - (a) Placement of all constants, variables, arrays, tables, and data in the lower 64 KB. This allows the use of MSP430 instructions with 16-bit addressing for all data accesses. No pointers with 20-bit addresses are needed.
 - (b) Placement of subroutine constants immediately after the subroutine code. This allows the use of the symbolic addressing mode with its 16-bit index to reach addresses within the range of PC +32 KB.

2. To use only MSP430X instructions: If the MSP430 instructions cannot be used. Two disadvantages of this method are the reduced speed due to the additional CPU cycles and the increased program space due to the necessary extension word for any double operand instruction.
3. Use the best fitting instruction where needed.

B.2 MSP430 Machine Instructions for the CPU

The complete MSP430 machine language set consists of 27 core instructions, each one associated to a unique op-code decoded by the CPU. The emulated instructions are instructions replaced automatically by the assembler with an equivalent core instruction, but have no opcode themselves. There is no code or performance penalty for using emulated instruction.

There are four MSP430 machine instruction formats:

- Dual-operand
- Single-operand
- Jump
- The `reti` instruction: 1300h

The `reti` group has only one instruction, `reti`. With very few exceptions, all single-operand and dual-operand instructions can be byte or word instructions by using `.B` or `.W` extensions. Byte instructions are used to access byte data, byte peripherals or the least significant byte of registers. Word instructions are used to access word data or word peripherals. If no extension is used, the instruction is by default a word instruction.

An instruction may consist of one, two or three words. The leading word is the instruction word, and the only one conveying information of the type of instruction and operands. The other words are either the immediate value of an immediate addressing word, or are values associated to addresses in memory. The bit fields for the instruction word are shown in Fig. B.1. The source (`src`) is defined by the `As` and `S-reg` fields while the destination (`dst`) by the `Ad` and `D-reg` fields. The fields are described as follows:

- As** Group of bits defining the addressing mode used for the source (`src`)
- S-reg** The working register used for the source (`src`), or as defined by `As`
- Ad** Bit defining the addressing mode used for the destination (`dst`)
- D-reg** The working register used for the destination (`dst`), or as defined by `Ad`.
- B/W** Byte or word operation:
 - 0: word operation
 - 1: byte operation

Notice that destination addresses are valid anywhere in the memory map. However, for an instruction that modifies the contents of the destination, the user must ensure the destination address is writable. For example, a masked-ROM location

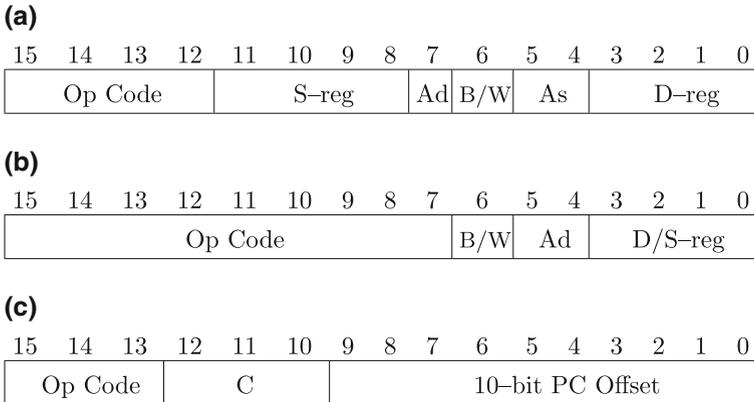


Fig. B.1 Word Structure for Machine Instruction words (a) double-operand instruction format (b) single-operand instruction format (c) Jump instruction format

Table B.1 Opcodes for dual-operand instructions

Hex Code (Bits 15-12)	Instruction	Hex Code (Bits 15-12)	Instruction
4	mov.w, mov.b	A	dadd.w, dadd.b
5	add.w, add.b	B	bit.w, bit.b
6	addc.w, addc.b	C	bic.w, bic.b
7	subc.w, subc.b	D	bis.w, bis.b
8	sub.w, sub.b	E	xor.w, xor.b
9	cmp.w, cmp.b	F	and.w, and.b

would be a valid destination address, but since the contents are not modifiable, the results of the instruction would be lost.

B.2.1 Operational Codes

From Fig. B.1 we see that the length of opcodes depend on the type of instruction. Dual-operand instructions, involving a destination and a source, show the most significant nibble as the opcode. The MSP430 CPU has 12 instructions falling in this category, all of which may be either word or byte type. These instructions together with the corresponding opcode are shown in Table B.1.

The most significant nibble equal to 1 is given to single operand instructions. The `reti = 1300h` instruction, which has no operands, also has this property. The opcode for a single operand instruction consists of the nine most significant bits, and some of these instructions are word type only. The instructions, together with their opcode, and the W/B bit, are shown in Table B.2.

Table B.2 OpCodes for single operand instructions

Bits 15-12	Bits 11-7	B/W bit	Instruction
0001	00000	0,1	<code>rrc, rrc.b</code>
0001	00001	0	<code>swpb*</code>
0001	00010	0,1	<code>rra, rra.b</code>
0001	00011	0	<code>sxt*</code>
0001	00100	0,1	<code>push, push.b</code>
0001	00101	0	<code>call*</code>

* Word instruction only

Table B.3 OpCodes for jump instructions

Bits 15-13	Bits 12-10	Instruction
001	000	<code>jnz/jneq</code>
001	001	<code>jz/jeq</code>
001	010	<code>jnc/jlo</code>
001	011	<code>jc/jhs</code>
001	100	<code>jn</code>
001	101	<code>jge</code>
001	110	<code>jl</code>
001	111	<code>jmp</code>

Finally, the jump instructions have six bits in their opcodes. For all cases, the three most significant bits are 001. This leaves three additional bits to define the jump, for a total of eight jump instructions. In the hex form, the most significant nibble corresponds to 2 or 3, depending on the C bits. The opcodes and jumps are shown in Table B.3.

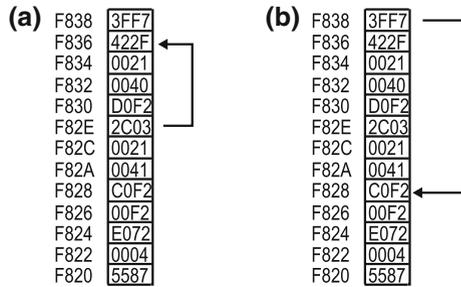
B.2.2 Operand Field in Jump Instructions

Jump instructions work in what is called an offset-relative mode. The execution of the jump consists in adding two times the signed offset value defined by the ten least significant bits of the instruction word. That is

$$PC \leftarrow PC + 2 \times (\text{ten-bit-PC-offset})$$

Here, the PC contents is that of the instruction address following the jump instruction. This is because execution phase occurs after the decode phase, when the PC is already updated to the new address. The ten-bit offset has a range from $-2^9 = -512$ to $2^9 - 1 = 511$ memory locations. Therefore, jumps can be done between -1024 and $+1022$ memory locations. For larger jumps, the emulated branch instruction should be used. The execution is illustrated in the following example.

Fig. B.2 Illustration for Jumps. **a** `jc : 2C03`, **b** `jmp : 3FF7`



Example B.1 Figure B.2 illustrates what happens when two jump instructions are executed. Case (a) corresponds to the machine language instruction 2C03h, while case (b) to 3FF7h.

In the first case, let us separate the opcode and the operand as 2C03 : **001011–0000000011**. The opcode corresponds to `jc / jhs` mnemonics. The ten bit operand is +3. Hence, when executed, if the carry flag is set, this instruction will add +6 to the PC contents. After decoding, PC is already pointing to the next location, in this case F830, so after execution it ends up with F836.

The instruction 3FF7 : **001111–1111110111** has the opcode corresponding to the unconditional jump and the operand stands for –9, which means it goes back 18 memory locations. The PC contents is F83A at that moment, so we have F83A + FFEE = 1F828 which yields memory location F828 discarding the carry.

B.2.3 Operand Definition from Instruction Word Fields

The two-bit *As* field, together with the 4-bit *S-Reg* field, define the source, with the combinations shown in Table B.4 for both one- and two-operands instructions. The destination operand is defined by the *Ad* bit and the *D-reg* group in dual-operand instructions as shown in Table B.5.

In the register fields, for both source and destination, registers R2 and R3 play a special role. When R2 appears in the source register field, and the *As* bits indicate register mode, then R2 refers to the Status Register itself. Similarly for the *D-register* field if *Ad* = 0. For other addressing modes, R2 is playing the role of constant generator, as explained later. R3 always plays the role of a constant generator, as explained in the next section.

B.2.4 Constant Generators

Registers R2 and R3 play a role of constant generators for particular cases of immediate mode values. Also, in absolute mode, &ADDR is equivalent to ADDR(R2),

Table B.4 Source definition

As1–As0 (Bits 5–4)	S-reg (Bits 11–8; 3–0) ¹	Comment
00	Reg. Number	Register mode
01	Reg. Number	Indexed Mode X(Rn)
01	0	Symbolic Mode ADDR. X value stored in the word following the instruction word, where X = PC - ADDR
01	2	Absolute mode &ADDR. SR takes value 0, and works as ADDR(SR) ² ADDR follows instruction word
10	Reg. Number	Indirect Register mode @Rn
11	Reg. Number	Indirect Autoincrement @Rn+
11	0	Immediate mode ³ #N N follows the instruction word

(1): Bits 11-8 in two operands, 3-0 in one operand instructions

(2): ADDR(SR) is invalid syntax in an instruction

(3): Technically, this is equivalent to @PC+

Table B.5 Destination definition

Ad (Bit 7)	D-reg (3-0)	Comment
0	Reg. Number	Register mode
1	Reg. Number	Indexed Mode X(Rn)
1	0	Symbolic Mode ADDR. X value stored in the word following the instruction word, where X = PC - ADDR
1	2	Absolute mode &ADDR ADDR is last word

where R2 takes the value 0. Constant generators are combined with As values in the way shown in Table B.6, corresponding to the immediate value in the respective line. The use of a constant generator saves a word in the machine instruction.

For example, `mov #1, R7` translates into **4317** (0010 **0011** 0001 0111), and `mov.b #4, R7` into **4267** (0010 **0011** 0110 0111).

The constant generators provide faster instructions, no special instructions are required, there is no additional code word for the six constants. The assembler uses the constant generator automatically if one of the six constants is used as an immediate source operand. When register R2 appears explicitly as an operand, it refers to the status register itself.

R3 as a source is equivalent to #0. R3 is valid as destination, but no value is stored. In fact, `nop` is emulated by `mov R3, R3`.

Table B.6 Immediate values and constant generators

Register (Bits 11-8)	As bits (Bits 5-4)	Equivalent operand
R2	00	SR in register mode*
R2	01	(0)**
R2	10	#4
R2	11	#8
R3	00	#0
R3	01	#1
R3	10	#2
R3	11	#-1 (=0xFFFF)

* R2 refers to the register SR itself, used in Register Mode.

** For absolute mode, as in &ADDR = ADDR(0)

B.3 Instruction Cycles

Remember that an instruction cycle covers three basic phases: fetch-decode-execute. This instruction cycle is also called a CPU cycle. By hardware, there is an internal clock in the CPU such that the number of cycles involved in one CPU cycle corresponds in time to one MCLK cycle. Hence, knowing the frequency of operation, and the number of CPU cycles required to execute one instruction, we can keep track of the timing required to execute loops, subroutines, and other tasks.

The number of CPU clock cycles required for an instruction depends on the instruction format and the addressing modes used - not the instruction itself. The number of clock cycles refers to the MCLK.

B.3.1 Interrupt and Reset Cycles

The CPU cycles involved in the processes of interrupts and requests are summarized in Table B.7.

Table B.7 Interrupt and reset cycles

Action	No. of Cycles
Return from interrupt (RETI)	5
Interrupt accepted	6
WDT reset	4
Reset (RST/NMI)	4

Table B.8 Interrupt and reset cycles

Addressing mode	No. of Cycles		
	RRA, RRC	PUSH	CALL
Register Rn	1	3	4
Indirect @Rn	3	4	4
Autoincrement Rn+	3	5	5
Immediate #N	N/A ¹	4	5
Indexed N(Rn)	4	5	5
Symbolic N	4	5	5
Absolute &N	4	5	5

(1) Do not use immediate mode in these instructions

B.3.2 Jump Instruction Cycles

All jump instructions require one code word, and take two CPU cycles to execute, regardless of whether the jump is taken or not.

B.3.3 Single Operand Instruction Cycles

B.3.4 Double Operand Instruction Cycles

In this case, the special situation in which the Program Counter (PC) register is used in destination must be considered apart. Thus, `mov R5, R6` and `mov R5, PC`, both with register addressing mode in the destination operand, have different cycle behavior.

Table B.9 Interrupt and reset cycles

Source addr. mode	Destination addr. mode	No. of cycles
Register Rn:		
	Register Rn	1
	Register PC	2
	Indexed N(Rn)	4
	Symbolic N	4
	Absolute &N	4
Indirect @Rn:		
	Register Rn	2
	Register PC	2
	Indexed N(Rn)	5
	Symbolic N	5
	Absolute &N	5
Autoincrement @Rn+ and Immediate #N:		
	Register Rn	2
	Register PC	3
	Indexed N(Rn)	5
	Symbolic N	5
	Absolute &N	5
Indexed N(Rn), Symbolic N, and Absolute &N:		
	Register Rn	3
	Register PC	3
	Indexed N(Rn)	6
	Symbolic N	6
	Absolute &N	6

Appendix C

Code Composer Introduction Tutorial

Materials

- CCS.
- MSP-EXP430G2 LaunchPad development board.

C.1 Introduction

Code Composer Studio (CCS) is TI's own compiler and debugging environment for the MSP430. Based on the Eclipse platform, CCS allows to leverage hundreds of plug-ins to accelerate code development. For all the upcoming experiments CCS will be used as the compiler, assembler, linker and code debugger for the MSP430 LaunchPad. The MSP-EXP430G2 LaunchPad is a complete MSP430 development tool. It includes all the hardware and software components to evaluate the MSP430 and develop a complete project in a convenient board. The module consists of two parts: a USB communications interface and chip programmer/debugger interface and an MSP430G2211/31 target board. The USB interface provides power to operate the ultra-low-power MSP430 so no external power supply is required to operate the module, making it portable and easy to use. It also generates the signal to program and debug code in the MSP430 memory. The MSP430 target board provides a fully functional MSP430 microcontroller with over 20 user accessible pins and several LED indicators.

C.2 Procedure

CCS Tutorial (These procedure assumes the CCS Integrated Development Environment (IDE) is already installed. If the CCS IDE is not installed, then go to the installation instructions at the end.) At the time of printing, the current CCS version is 4.2.1.00004.

1. On the workstation go to 'Start > All Programs > Texas Instruments > Code Composer Studio'. Additional letters and numbers will follow after the word Studio, depending on the installed version.
2. If this is the first time using CCS, it will ask a location (workspace) to store your project files. You may choose the path shown. Alternatively, you may choose to use any other location as your workspace. Regardless of your choice, please make sure you are able to recognize this directory in the future. We will assume the workplace is at installed path Workspace.
3. To create a new project: select 'File > New > CCS Project'.
4. Name your project "EXP1". Use the default location (in the current workspace). Continue with default settings by clicking 'Next' until the 'Project Settings' page appears. If you get the 'Project Type' page before getting to the 'Project Settings' page, make sure that you choose MSP430 in the pull down menu.
5. Select "MSP430G2231" under 'Device Variant' and click 'Next'.
6. Make sure that you choose "Empty Assembly-only Project" and then click 'Finish'. If you have clicked 'Finish' instead of 'Next' in the previous step, you would have chosen the option that expects you to use C in your project and this would have caused problems later on.
7. Look inside the 'C/C++ Project' tab to make sure that EXP1 is the active project. If it is not, right click on the name of your project and choose 'Set as Active Project'. Now click 'File > New—Source File'. Name your source file "FlashLed.asm" under 'Souce File:'
8. Type the code shown below in Listing C.1 as is, except for the line numbers, into the source window and then save the work done by clicking 'File > Save'.

Listing C.1 Your first assembly language program.

```

1      .cdecls C , LIST , "msp430g2231.h"
2      ; - - - - -
3      .text ; Progam Start
4      ; - - - - -
5  RESET   mov     # 0280h , SP ; Initialize stackpointer
6  StopWDT mov     # WDTPW+WDTHOLD , & WDTCTL ; Stop WDT
7  SetupP1 bis.b   # 1,& P1DIR ; P1.0 as output
8  Main    xor.b   # 1,& P1OUT ; Toggle P1.0
9  Wait    mov     # 50000 , R15 ; Delay to R15
10 L1      dec     R15 ; Decrement R15
11        jnz    L1 ; Delay over?
12        jmp    Main ; Again
13 ; - - - - -
14 ;          Interrupt Vectors
15 ; - - - - -

```

```

16         .sect      ".reset"      ; MSP430 RESET Vector
17         .short    RESET          ;
18         .end

```

9. Connect the LaunchPad to the computer via the USB cable provided with it. If this is the first time the LaunchPad is connected, you will notice that Windows will install the appropriate driver. This driver is supplied as part of CCS. The rest of this experiment relies on the assumption that the driver is properly installed.
10. You can now click 'Target > Debug Active Project'. The Progress Information page is displayed while the code downloads. Once the download is completed the debug perspective opens automatically.
11. If there are no errors, you can now run your project by clicking Target—Run or by hitting the F8 key. If there are any errors, then they must be syntax errors as opposed to logic errors or programming errors. After typing ";" on any line, the assembler ignores the rest of the line, not the rest of the program. This means that it is not important if you have made a typographical error after a ";". Review each line to make sure that the code entered is as shown. The above code has been properly tested and it was run to make sure that the steps shown here will produce the same results for you.
12. The red LED should be toggling on and off in the MSP-EXP430G2 LaunchPad. If it does not, then you should 'debug' your program to find the errors.

C.3 Exercises

1. Replace the #50000 with #100000. Save your changes by clicking 'File > Save'. You can now click 'Target > Debug Active Project'. The 'Progress Information' page is displayed. Now hit the F8 key to run your program. What change did you observe on the red LED toggling frequency?

Note that we made the changes on the FlashLed.asm file in the computer while the program was running on the LaunchPad. We could have also terminated the program by clicking 'Target > Terminate All' or Ctrl+Alt+T. We could have also halted the program by clicking 'Target > Halt' or Shift+F8. After terminating or halting the program, we can also make the desired changes and then click 'Target > Debug Active Project' and hit F8 with similar results. If you run into problems we advise you to exit CCS and then continue with your work.

2. Repeat the above exercise replacing the #50000 or #100000 with #2500. While your project is running, click 'View > Registers'. A tab should appear with the label "Registers (1)". Double click to expand it. Now click "Core Registers". Do the same with "Special_Function". You will probably see "Unable to read" under the column "Value". Halt your project by clicking 'Target > Halt' or pressing Shift+F8. You should now see some hexadecimal numbers under the "Value" column. Go ahead and change the value for register R5 by clicking the value for

R5. Now enter decimal number 256 and hit the Enter key. You should see that the new value is 0x0100, which is decimal 256 written in hexadecimal notation. Now change it to 0xabcd. You should see that it changed the contents to 0xABCD. Hit F8 to run the project. We just showed you a way to examine and change the contents of the MSP430 registers. If you need to examine or change the contents of other parts of the chip, you just need to follow a similar procedure. At this time we want to show you another feature included with CCS. While your program is running, click 'View > Disassembly'. You should see that a new window opens in CCS with the name Disassembly. At this time the window should be empty. Now halt your project and you should see something like this in the Disassembly Tab:

```

1          RESET , .text , _text , $.. / FlashLed . asm : 25 : 44$:
2 0xF800: 4031 0280      MOV.W      #0x0280 , SP
3          StopWDT:
4 0xF804: 40B2 5A80 0120    MOV.W      #0x5a80 , & Watchdog_ ...
5          SetupP1:
6 0xF80A: D3D2 0022      BIS.B      #1 , & Port_1_2_P1DIR
7 0xF80E: D3C2 0021      BIS.B      #0 , & Port_1_2_P1OUT
8          Main:
9 0xF812: E3D2 0021      XOR.B      #1 , & Port_1_2_P1OUT
10         Wait:
11 0xF816: 403F 5000      MOV.W      #0x5000 ,R15
12         L1:
13 0xF81A: 831F      DEC.W      R15
14 0xF81C: 23FE      JNE        (L1)
15 0xF81E: E0F2 0041 0021    XOR.B      #0x0041 , & Port_1_2_P1OUT
16 0xF824: 403F 5000      MOV.W      #0x5000 , R15
17         L2:
18 0xF828: 831F      DEC.W      R15
19 0xF82A: 23FE      JNE (L2)
20 0xF82C: E0F2 0081 0021    XOR.B      #0x0081 , & Port_1_2_P1OUT
21 0xF832: 3FEF      JMP        (Main)

```

You probably recognize the instructions in capital letters since you entered them when you were writing the code. There are, however, several things that were added to your program. For example, the second line, which is actually the first instruction, is displayed:

```

1 0xF800: 4031 0280      MOV.W      #0x0280 , SP

```

At the beginning of the above line you see 0xF800: 4031 0280. The string of characters 0xF800 is indicating the address of the location in memory whose content is 4031 0280. This two character strings are the machine language version of the assembly language instruction MOV.W #0x0280,SP. The first of these two strings, 4031, indicates that this is a MOV instruction working on two bytes or 16 bits at a time, while the second string, #0x0280, specifies the source operand for the MOV instruction.

If you only leave the program memory addresses and their content we would see this

```

1  0xF800  RESET , . text , _text , $../FlashLed . asm : 25 : 43$
2  0xF800  4031 0280
3  0xF804  40B2 5A80 0120
4  0xF80A  D3D2 0022
5  0xF80E  D3C2 0021
6  0xF812  E3D2 0021
7  0xF816  403F 5000
8  0xF81A  831F
9  0xF81C  23FE
10 0xF81E  EOF2 0041 0021
11 0xF824  403F 5000
12 0xF828  831F
13 0xF82A  23FE
14 0xF82C  EOF2 0081 0021
15 0xF832  3FEF

```

Now click 'View > Memory' and, when the memory window opens, type 0xF800 in the textbox where it says 'Enter location here'. You should see a display similar to the above. At this time you should not be concerned with the actual meaning of the assembly language instruction or its machine language version. Our intention at this time was to show you that with CCS you can see the program memory and its actual content. This will come in handy in the future.

3. We would like now to toggle the red LED and the green LED at the same time. The only changes we need to make are the following: replace the #1 with #0x41 in the two lines where the #1 appears, i.e. lines 7 and 8 in Listing C.1. Now click 'Target > Debug Active Project' and then hit F8 (or 'Target > Run') to run your project.
4. For this part of the exercise we will assume that the red LED is toggling on and off. Locate connector J5. Now grab jumper P1.0 with your fingers and pull it out of the connector. The red LED should be off because no power is delivered to it although the program is still running. Replace the jumper and the red LED should once again begin to toggle on and off. If the green LED is toggling on and off, you could do the same only this time you need to work with jumper P1.6.
5. Another exercise is to turn each LED in turn. One way to accomplish this is as follows. Insert

```
mov.b    #0,&P1OUT    ; both LEDs off
```

after line 7 and before line 8. The above line should now be line 8.

Now insert the following instructions

```

xor.b    #01000001b,&P1OUT; P1.0 off, P1.6 on
mov.w    #050000,R15
L2      dec.w    R15
        jnz     L2
xor.b    #01000001b,&P1OUT ; P1.0 on, P1.6 off

```

just before the new line 13, i.e. just before

```
jmp Main
```

Your code should now look like Listing C.2:

Listing C.2 Modified code for turning each LED in turn.

```

1      .cdecls C,LIST, "msp430g2231.h"
2      ;-----
3      .text      ; Program Start
4      ;-----
5  RESET      mov.w   #0280h , SP ; Initialize stackpointer
6  StopWDT    mov.w   #WDTPW+WDTHOLD , & WDTCTL ; Stop WDT
7  SetupP1    bis.b   #0x41,&P1DIR ; P1.0, P1.6 as output
8            mov.b   #0 , & P1OUT ; both LEDs off
9  Main       xor.b   #1 , & P1OUT ; Toggle P1.0
10 Wait      mov.w   #050000 , R15 ; Delay to R15
11 L1         dec.w   R15      ; Decrement R15
12           jnz     L1        ; Delay over?
13           xor.b   #0x41 , & P1OUT ; P1.0 off , P1.6 on
14           mov.w   #050000 , R15
15 L2         dec.w   R15
16           jnz     L2
17           xor.b   #1 , & P1OUT ; P1.0 on, P1.6 off
18           jmp     Main ; Again
19 ;-----
20 ;           Interrupt Vectors
21 ;-----
22           .sect   ".reset" ; MSP430 RESET Vector
23           .short  RESET ;
24           .end

```

C.4 Installing CCS

1. Obtain the CCS setup application. You should be able to obtain the CCS application at <http://www.ti.com>. If it is in a compressed format, then you should extract the files to a destination of your choice.
2. Double click the CCS setup application. The install wizard displays progress information.
3. Follow the wizard prompts during the installation.
4. After reading the license agreement, accept it if you still want to install CCS and then click 'Next'.
5. If this is the first time CCS will be installed on your computer, choose the default path settings for the installation location. Click 'Next'. If you have already installed CCS on your computer, make sure you choose a different path for this installation.
6. When prompted for the product configuration, choose the 'MSP430-only Core tools' and then click 'Next'.
7. The setup wizard will display the components it is about to install of the install. Click 'Next'.
8. Now you will be shown a summary of the changes the setup wizard is about to make. Click 'Next' if you agree with the information.

9. The wizard displays progress information, wait until the installation is completed.
10. Click 'Finish' when prompted.

The installation instructions shown above are general in nature and assume you are using the Windows 7 operating system (OS). Depending on your OS and the CCS version, the setup wizard may behave differently. You should make sure that your computer and OS meet the requirements for installing CCS. You should also make sure that your OS is aware of your computer hardware such as the chipset. For example, CCS may not work properly on a computer built for Windows Vista or Windows 7 if the computer is running an older version of Windows like XP, because Windows XP may not have the appropriate driver for the machine. You should also be aware that, on some operating systems, like Windows 7, you may have to make the installation from the Administrator account.

Appendix D

CPUX: An Overview of the Extended MSP430X CPU

D.1 Introduction

The MSP430X (CPU_x) is a 20-bit version of Texas Instruments MSP430 microcontroller. It was introduced with the second generation of MSP430 chips, i.e. with the MSP430x4xx family, and it has been present with all subsequent versions, except for the MSP430x1xx.

CPU_x's 20-bit address bus, see Fig. D.1, allows to reach the 1-MB address space without paging. Remember that a traditional MSP430 is a 16-bit processor both at its address and data buses and can reach a 64-KB address space.

CPU_x is backward compatible with the MSP430 CPU and it can address bytes, 16-bit words, and 20-bit words. It maintains its orthogonal RISC architecture allowing any CPU register to be used as an operand. Several instructions have been extended for 20 bit operation. However, using a prefix any instruction can be extended to 20 bit. CPU_x has fewer interrupt overhead cycles and fewer instruction cycles in some cases than the MSP430 CPU.

As shown in Table D.1, there are several instructions that were extended to take advantage of the increased address space, namely: MOVA, CALLA, ADDA, SUBA, CMPA. ADDA, SUBA, and CMPA are restricted to the immediate and register addressing modes only, i.e. you cannot use the rest of the addressing modes with these three (3) instructions.

On the other hand, as summarized in Table D.2 there are several instructions for performing multi-bit shifts (1, 2, 3, or 4 bits): RRCM, RRAM, RLAM, RRUM. You can also push or pop several registers with the instructions PUSHM and POPM.

By using an additional word of op-code called an extension word, all addresses, indexes, and immediate values are extended to 20 bit. For example, BISX sets bits in the destination word that are set in the source word, BISX.B sets bits in the destination byte that are set in the source byte, and BISX.A sets bits in the destination address-work that are set in the source address-word.

You can do similar operations with ADDX, ADDCX, ANDX, BICX, BISX, BITX, CMPX, DADDX, MOVX, POPM, PUSHM, PUSHX, RLAM, RRAM,

Fig. D.1 MSP430X (CPUX) Block Diagram. *Courtesy of Texas Instruments, Inc.*

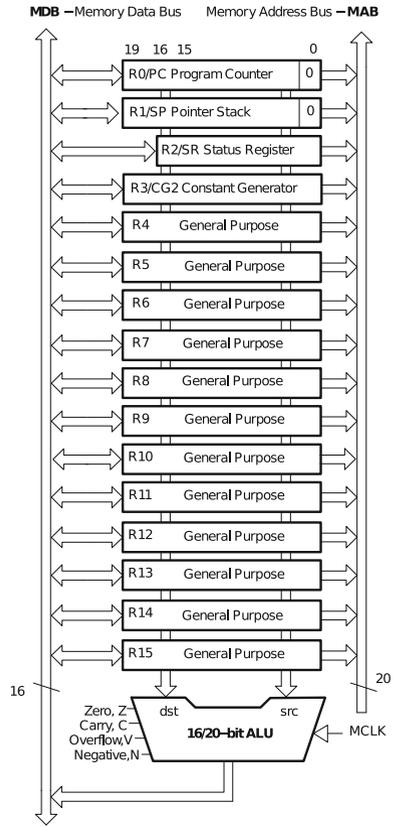


Table D.1 MSP430X new instructions

	Instruction	Restriction
1	MOVA	
2	CALLA	
3	ADDA	Only immediate and register addressing modes.
4	SUBA	Only immediate and register addressing modes.
5	CMPA	Only immediate and register addressing modes.

RRAX, RRCM, RRCX, RRUM, RRUX, SUBX, SUBCX, SWPBX, SXTX, and XORX.

Table D.3 shows the instructions that have been extended in the MSP430X and the original MSP430 instructions.

Table D.2 MSP430X Multibit instructions

	Instruction	Description
1	RRCM(.W/.A)	Rotate right n bits through carry.
2	RRAM(.W/.A)	Rotate right n bits arithmetically.
3	RLAM(.W/.A)	Rotate left n bits arithmetically.
4	RRUM(.W,.A)	Rotate right n bits unsigned.
5	PUSHM(.W/.A)	Push n 16 or 20 bit registers onto the stack.
6	POPM(.W/.A)	Pop n 16 or 20 bit register from the stack.

Table D.3 MSP430X Extended and original instructions

	Extended instruction	Original CPU instruction	Extended instruction	Original CPU instruction
1	ADDX(.B/.W/.A)	ADD	PUSHX(.B/.W/.A)	PUSH
2	ADDCX(.B/.W/.A)	ADDC	RRAX(.B/.W/.A)	RRA
3	ANDX(.B/.W/.A)	AND	RRCX(.B/.W/.A)	RRC
4	BICX(.B/.W/.A)	BIC	SUBX(.B/.W/.A)	SUB
5	BISX(.B/.W/.A)	BIS	SUBCX(.B/.W/.A)	SUBC
6	BITX(.B/.W/.A)	BIT	SWPBX(.W/.A)	SWPB
7	CMPX(.B/.W/.A)	CMP	SXTX(.W/.A) ¹	SXT
8	DADDX(.B/.W/.A)	DADD	TSTX(.B/.W/.A)	TST
9	MOVX(.B/.W/.A)	MOV	XORX(.B/.W/.A)	XOR
10	POPX(.B/.W/.A)	POP		

1 The operation of the extended instruction is not completely similar to that of the original MSP430 one

D.2 Differences Between the MSP430X and MSP430

1. Address bus

- (a) MSP430 has a 16 bit address bus.
- (b) MSP430X has a 20 bit address bus.

2. Address space

- (a) MSP430 can access up to 2^{20} locations.
- (b) MSP430X can access up to 2^{16} locations.

3. Memory accesses

- (a) MSP430 memory accesses are 8 and 16 bits.
- (b) MSP430X memory accesses are 8, 16, and 20 bits.

4. Size of registers

- (a) MSP430 registers are 16 bit long.
- (b) MSP430X registers are 20 bit long, except for the SR register which is 16 bit long.

5. Register Accesses

- (a) MSP430 register accesses are 8 and 16 bits.
- (b) MSP430X register accesses are 8, 16, and 20 bits.

6. R/W RAM

- (a) MSP430 starts at location 0x0200.
- (b) MSP430X starting location varies.

7. Interrupt Vector Table

- (a) MSP430: 0xFFFF - 0xFFC0.
- (b) MSP430X: 0xFFFF - 0xFF80.

8. Constants generated with constant generators

- (a) MSP430: (0) for absolute mode, 0x0000, 0x0001, 0x0002, 0x0004, 0x0008, and -1 as 0xFFFF.
- (b) MSP430X: (0) for absolute mode, 0x00000, 0x00001, 0x00002, 0x00004, 0x00008, and -1 as 0xFF, 0xFFFF, and 0xFFFFF.

9. Interrupt latency

- (a) MSP430: 6 cycles.
- (b) MSP430X: 5 cycles.

10. Interrupt return

- (a) MSP430: 5 cycles.
- (b) MSP430X: 3 cycles.

11. Four (4) bit extension word in opcode

- (a) MSP430: Does not apply.
- (b) MSP430X: Present.

12. Can extend all addresses, indexes, and immediate values to 20 bit

- (a) MSP430: Does not apply.
- (b) MSP430X: True.

13. Multibit (1, 2, 3, or 4 bits) shift instructions

- (a) MSP430: Does not apply.
- (b) MSP430X: True.

14. Instructions for pushing or popping several (1 to 16) registers

- (a) MSP430: Does not apply.
- (b) MSP430X: Present.

15. Can perform direct access and branching throughout entire memory range without paging

- (a) MSP430: Does not apply.
 - (b) MSP430X: True.
16. Instructions length
- (a) MSP430: 1, 2, or 3 16-bit words.
 - (b) MSP430X: 1, 2, 3, or 4 16-bit words.
17. BR and CALL reset upper four PC bits to 0
- (a) MSP430: Does not apply.
 - (b) MSP430X: True.
18. Appends bits 19 - 16 to stored SR on stack during interrupt
- (a) MSP430: Does not apply.
 - (b) MSP430X: True.
19. Byte-write to register clears (except SXT instruction)
- (a) MSP430: bits 15 - 8.
 - (b) MSP430X: bits 19 - 8.
20. Word-write to register clears (except SXT instruction)
- (a) MSP430: Does not apply.
 - (b) MSP430X: bits 19 - 16.
21. Register to Address-word (.A) clears
- (a) MSP430: Does not apply.
 - (b) MSP430X: bits 15 - 4 on second 16 bit word.
22. Address-word to Register
- (a) MSP430: Does not apply.
 - (b) MSP430X: bits 15 - 4 on second 16 bit word not used.
23. Instructions limited to immediate and register addressing modes
- (a) MSP430: Does not apply.
 - (b) MSP430X: ADDA, SUBA, and CMPA
24. Effect of X instructions
- (a) MSP430: Does not apply.
 - (b) MSP430X:
 - i. Reduced speed due to additional CPU cycles.
 - ii. Increased space due to extension word for double operand instructions.

Appendix E

Copyright Notices

This book contains code examples and figures that were either developed as original material for this book or obtained from external open sources available elsewhere. Materials obtained from such sources are identified next to their instances and their source listed in the Bibliography section of this book. This appendix reproduces the copyright notices of the original distribution of those materials obtained from external sources. The authors of this book have included such examples and figures as illustrative instances of the subjects discussed in each case. Although the authors have made every effort to verify their correctness, these materials are provided “as is”. Any express or implied warranties, including, but not limited to, the implied warranties of fitness for any particular purpose are disclaimed. Under no circumstance or event shall the authors or the copyright owners be liable for any direct, indirect, incidental, exemplary, or consequential damages arising from the use of these materials.

E.1 Copyright Notice for Code Examples from Texas Instruments

```
; --COPYRIGHT--,BSD_EX
; Copyright (c) 2012, Texas Instruments Incorporated
; All rights reserved.
;
; Redistribution and use in source and binary forms, with or without
; modification, are permitted provided that the following conditions
; are met:
;
; * Redistributions of source code must retain the above copyright
;   notice, this list of conditions and the following disclaimer.
;
; * Redistributions in binary form must reproduce the above copyright
;   notice, this list of conditions and the following disclaimer in the
;   documentation and/or other materials provided with the distribution.
;
; * Neither the name of Texas Instruments Incorporated nor the names of
;   its contributors may be used to endorse or promote products derived
;   from this software without specific prior written permission.
```

```

;
; THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
; AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
; THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
; PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
; CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
; EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
; PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
; WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
; OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
; EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
;
; *****
;
; MSP430 CODE EXAMPLE DISCLAIMER
;
; MSP430 code examples are self-contained low-level programs that typically
; demonstrate a single peripheral function or device feature in a highly
; concise manner. For this the code may rely on the device's power-on default
; register values and settings such as the clock configuration and care must
; be taken when combining code from several examples to avoid potential side
; effects. Also see www.ti.com/grace for a GUI- and www.ti.com/msp430ware
; for an API functional library-approach to peripheral configuration.
;
; --/COPYRIGHT--

```

References

1. Sale, T. (2000). Lorentz Ciphers and the Colossus. *Technical report, Bletchley park museum*, Retrieved June, 2008, from <http://www.codesandciphers.org.uk/lorenz>.
2. Pultorak, J. (2004). Block I Apollo Guidance Computer (AGC): How to build one in your basement. *Technical report, NASA office of logic design*, Retrieved June, 2008, from http://www.klabs.org/history/build_agc/.
3. Daniels, R. G. (1996). A participant's perspective. *IEEE Micro*, 16(6), 21–31.
4. Holt, R. M. (1998). Architecture of a microprocessor. Retrieved June, 2008, from <http://www.microcomputerhistory.com/fl4paper.htm>.
5. Penumuchu, C. V. (2007). *Simple real-time operating system: A kernel inside view for a beginner*. Victoria, VC, Canada: Trafford Publishing, Inc.
6. Koopman, P. (1996). Embedded system design issues (the rest of the story). In *Proceedings of the International Conference on Computer Design (ICCD 96)*, (pp. 310–317), October 1996.
7. Scheffer, L. (2006). *EDA for IC system design, verification, and testing*. Boca raton, FL: CRC Press.
8. Madisetti, V. K. (1996). Rapid digital system prototyping: current practice, future challenges. *IEEE Design and Test of Computers*, 13(3):12–22.
9. Peckol, J. K. (2008). *Embedded systems: A contemporary design tool*. Hoboken, NJ: John Wiley & Sons, Inc.
10. Roth, K., & McKenney, K. (2007). Energy Consumption by Consumer Electronics in U.S. Residences. Technical report, Consumer Electronics Association/TIAX LLC.
11. Malik, S., Tiwari, V., & Wolfe, A. (2001). Power estimation in embedded systems: A hardware/software codesign approach. *Readings in hardware/software codesign* (pp. 249–258). Norwell, MA: Kluwer Academic Publishers.
12. Kmetovicz, R. E. (1992). *New product development: Design and analysis*. New York, NY: John Wiley & Sons, Inc.
13. Lofgren, J. D. (2004). A generic test and maintenance node for embedded system test. In *Proceedings of the International Test Conference 2004*, (pp. 143–153). October 2004.
14. Lindvall, M., Komi-Sirviö, S., Costa, P., & Seaman, C. (2003). A State of the Art Report: Embedded Software Maintenance. Technical Report 0704–0188, Data and Analysis Center for Software, 775 Daedlian Dr., Rome NY, January 2003.
15. Mano, M. M., & Ciletti, M. D. (2006). *Digital design* (4th ed.). New York, NY: Pearson College Div.
16. Daniels, W. L. (2002). *Fundamentals of embeded software: where C and assembly meet*. Upper Saddle River, NJ: Prentice Hall Inc.
17. Parhami, B. (2000). *Computer arithmetic: Algorithms and hardware design*. New York, NY: Oxford University Press.

18. IEEE Computer Society. IEEE ANSI/IEEE Standard 754–1985, Standard for Binary Floating Point Arithmetic. IEEE Society, 1985.
19. Fletcher, W. I. (1980). *An engineering approach to digital design*. Englewood Cliffs, N.J.: Prentice Hall, Inc.
20. Stallings, W. (2009). *Computer organization and architecture*. Upper Saddle River, NJ: Prentice Hall Inc.
21. Korpela, J. K. (2006). *Unicode explained*. Sebastopol, CA: O'Reilly Media, Inc.
22. Davies, J. H. (2008). *MSP430 Microcontroller basics*. Burlington, MA: Elsevier, Inc. Newnes.
23. Inc. Texas Instruments. (2012). Msp430x2xx family user's guide, slau144i. Electronic, Texas Instruments Inc, Post Office Box 655303 Dallas, Texas 75265, January 2012.
24. Inc. Texas Instruments. (2011). Msp430x5xx/msp430x6xx family user's guide, slau208i. Technical report, Texas Instruments, Inc., Post Office Box 655303 Dallas, Texas 75265, September 2011.
25. Inc. Texas Instruments. (2006). Msp430x1xx family user's guide, slau049f. Technical report, Texas Instruments, Inc., Post Office Box 655303 Dallas, Texas 75265, 2006.
26. Inc. Texas Instruments. (2002). Msp430x3xx family user's guide, slau012a. <http://www.ti.com/lit/ug/slau012a/slau012a.pdf>.
27. Inc. Texas Instruments. (2010). Msp430x4xx family user's guide, slau056j. Technical report, Texas Instruments, Inc., Post Office Box 655303 Dallas, Texas 75265, June 2010.
28. Kelley, A. I., & Pohl, I. (1990). *A book on C: Programming in C*. Boston: Addison-Wesley.
29. Msp430 optimizing c/c++ compiler v 3.0 user's guide, march 2008, 2008.
30. Inc. IAR Systems. (2006). Msp430 iar c/c++ compiler reference guide for texas instruments' msp430 microcontroller family. Technical report.
31. Kubes, D. (2012). The 5-minute guide to c pointers. Retrieved August, 2012, from <http://denniskubes.com/2012/08/16/the-5-minute-guide-to-c-pointers/>.
32. Inc. Texas Instruments. (2012). Msp430x2xx family user's guide, slau144i. Retrieved December, 2004, from <http://www.ti.com/lit/ug/slau144i/slau144i.pdf>. (Revised January 2012).
33. Dannenberg, A. (2006). msp430x22x4_ta_08. Retrieved April, 2006, from, <http://www.ti.com/lit/zip/slac123>.
34. Iar c/c++ compiler reference guide for texas instruments+ msp430 microcontroller family, march 2010, 2010.
35. Buccini, M., & Westlund, L. (2005). Msp430f20xx demo - timer_a, toggle p1.0, ccr0 cont. mode isr, dco smclk. Retrieved October, 2005, from <http://www.ti.com/lit/zip/slac080i>.
36. Westlund, L. (2006). Msp430f20x2 demo - adc10, sample a1, avcc ref, set p1.0 if > 0.5*avcc. Retrieved May, 2006, from <http://www.ti.com/lit/zip/slac080i>.
37. Buccini, M. & Westlund, L. (2005). Msp430f20xx demo - usicnt used as a one-shot timer function, dco smclk. Retrieved September, 2005, from <http://www.ti.com/lit/zip/slac080i>.
38. Inc. Energizer Holdings. Energizer 522 alkaline 9v battery, form no. ebc - 1108k. <http://data.energizer.com/PDFs/522.pdf>.
39. Kundert, K. (2012). Power supply noise reduction. Retrieved January, 2004, from <http://www.designers-guide.org>. (Last retrieved November 2012).
40. National Semiconductor (Now Texas Instruments). (2012). Lp3470: Timy power-on reset circuit. Retrieved September, 2009, from <http://www.ti.com/lit/gpn/lp3470>. (Last retrieved December 2012).
41. Dannenberg, A. (2005). Msp430f22x4 toggle p1.0 demo. Retrieved April, 2005, from <http://www.ti.com/lit/zip/slac123>.
42. Atmel Corporation. (2012). Doc9108: Microcontroller in a harsh environment. Retrieved October, 2007, from <http://www.atmel.com/Images/doc9108.pdf>. (Last retrieved December 2012).
43. Kleidermacher, D. (2005). Minimizing interrupt response time. *Information Quarterly*, 4(1), 52–54.
44. Keith, Q. (2006). Msp430 software coding techniques. Technical report, Texas Instruments, 2006. Revised version, August 2006.
45. Inc. Texas Instruments. (2012). Msp430 low cost pinosc capacitive touch overview. Retrieved September, 2011, from http://processors.wiki.ti.com/index.php/MSP430_Low_Cost_PinOsc_Capacitive_Touch_Keypad. (Last retrieved December 2012).

46. Bierl, L. (2002). Interfacing the 3-v msp430 to 5-v circuits (slla148). Retrieved October, 2002, from <http://www.ti.com/litv/pdf/slla148>.
47. Inc. Maxim Integrated. (2012). Logic-level translation (an3007). Retrieved July, 2004, from <http://pdfserv.maximintegrated.com/en/an/AN3007.pdf>. (Last retrieved November 2012).
48. Inc. Texas Instruments. (2011). Msp430f22x2/f22x4 mixed signal microcontroller. Retrieved July, 2011, from <http://www.ti.com/docs/prod/folders/print/msp430f2274.html>.
49. Fornaciari, W., Gubiani, P., Sciuto, D., & Silvano, C. (June 1988). Power estimation of embedded systems: A hardware/software codesign approach. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 6(2), 266–275.
50. Seshasayee, N. (2012). Understanding thermal dissipation and design of a heatsink (slva462). Retrieved May, 2011, from <http://www.ti.com/litv/pdf/slva462>. (Last retrieved November 2012).
51. Bachman, P., & Haiduk, R. (2011). The effect of forced air cooling on heat sink thermal ratings. Retrieved July, 2011, from http://www.crydom.com/en/Tech/HS_WP_FA.pdf.
52. Ganssle, J. G. (2008). A guide to debouncing. Retrieved June, 2008, from <http://www.ganssle.com/debouncing.htm> The Ganssle Group PO Box 38346 Baltimore, MD 21231.
53. Slater, M. (1989). *Microprocessor-based design: A comprehensive guide to effective hardware design*. Englewood Cliffs, NJ: Prentice Hall, Inc.
54. Kingbright USA. Super-bright red three digit numeric display. On the World Wide Web. March 2011.
55. Fairchild Semiconductor Co. 2n3904 / mmbt3904 / pzt3904 npn general purpose amplifier. On the World Wide Web, October 2011.
56. National Semiconductor (Now Texas Instruments). Mm5452/mm5453 liquid crystal display drivers. Retrieved December, 2008, from <http://www.ti.com/product/mm5452>.
57. Ltd. Hitachi. Dot matrix liquid crystal display controller/driver. Retrieved September, 1999, from <http://www.semiconductor.hitachi.com/>.
58. Inc. Texas Instruments. (2010). Msp430x4xx family user's guide. Retrieved January, 2010, from <http://www.ti.com/litv/pdf/slau056j>.
59. Inc. Texas Instruments. (2012). Msp430x5xx/msp430x6xx family user's guide. Retrieved August, 2012, from <http://www.ti.com/litv/pdf/slau208k>.
60. Luxeon Star LEDs. Sr-05rebel 10mm square led assembly. Retrieved September, 2012, from <http://www.luxeonstar.com//v/vspfiles/downloadables/sr-05.pdf>.
61. Inc. Texas Instruments. (2012). Msp430f11x2, msp430f12x2 mixed signal microcontroller (rev. d). Retrieved August, 2004, from <http://www.ti.com/product/msp430f1222>. (Last downloaded November 2012).
62. ON Semiconductor. Npn small-signal darlington transistor data sheet. Retrieved January, 2012, from <http://www.onsemi.com/pub/Collateral/BSP52T1-D.PDF>.
63. Inc. Texas Instruments. (1997). Uln2803a darlington transistor array (slrs049c). Retrieved February, 1997, from <http://www.ti.com/lit/ds/symlink/uln2803a.pdf>.
64. Hughes, A. (2005). *Electric motors and drives: Fundamentals, types and applications* (3rd edn.). New York: Newones.
65. Condit, R., & Jones, D. W. (2012). An907: Stepping motors fundamentals. Retrieved February, 2004, from <http://ww1.microchip.com/downloads/en/AppNotes/00907a.pdf>. (Last retrieved December 2012).
66. Inc. Motorola. (2012). Mc 3479 stepper motor driver. Retrieved July, 1996, from <http://www.futurlec.com/Motorola/MC3479P.shtml>. (last retrieved December 2012).
67. Inc. Texas Instruments. (2010). Drv8811 stepper motor controller ic. Retrieved May, 2010, from <http://www.ti.com/lit/ds/symlink/drv8811.pdf>.
68. Soltero, M., Zhang, J., Cockril, C., Zhan, Z., Kinnaird, C., & Kugelstadt, T. (2012). Rs-422 and rs-485 standards overview and system configurations (slla070d). Retrieved May, 2010, from <http://focus.ti.com/lit/an/slla070d/slla070d.pdf>. (Last retrieved December 2012).
69. Texas Instruments' Application Report. Interface circuits for tia/eia-485 (rs-485) (slla036d). Retrieved August, 2008, from <http://www.ti.com/lit/an/slla036d/slla036d.pdf>. (Last retrieved December 2012).

70. Texas Instruments Technical Staff. Comparing bus solutions (slla067b). Retrieved October, 2009, from <http://www.ti.com/lit/an/slla067b/slla067b.pdf>. (Last retrieved December 2012).
71. Universal serial bus specification, revision 2.0. Retrieved April, 2000, from <http://www.usb.org>. (Last retrieved December 2012).
72. Universal serial bus 3.0 specification, revision 1.0. Retrieved June, 2011, from <http://www.usb.org>. (Last retrieved December 2012).
73. Kollman, R. & Betten, J. (2012). Powering electronics from the usb port. Retrieved April, 2002, from <http://www.ti.com/sc/analogapps>. (Last retrieved December 2012).
74. NXP Semiconductors. I2c-bus specification and user manual - rev. 5 (um10204). Retrieved October, 2012, from http://www.nxp.com/documents/user_manual/UM10204.pdf. (Last retrieved January 2013).
75. Buccini, M., & Morton, G. (2005). Msp-fet430p140 demo - usart0, uart 115200 echo isr, hf xtal aclk. Retrieved May, 2005, from <http://www.ti.com/lit/zip/slac015>.
76. Texas Instruments Technical Staff. Tlc549c 8-bit analog-to-digital converters with serial control - slas067c. Retrieved September, 1996, from <http://www.ti.com/lit/ds/symlink/tlc549.pdf>.
77. Buccini, M. & Westlund, L. Msp430f20x2/3 demo - usi spi interface to tlc549 8-bit adc. Retrieved October, 2005, from <http://www.ti.com/lit/zip/slac149f>.
78. Nisarga, B. (2007). Msp430x24x demo - usci_a0, 9600 uart echo isr, dco smclk. Retrieved September, 2007, from <http://www.ti.com/lit/zip/slac149f>.
79. Nisarga, B. (2007). Msp430x24x demo - usci_a0, 9600 uart, smclk, lpm0, echo with over-sampling. Retrieved September, 2007, from <http://www.ti.com/lit/zip/slac149f>.
80. Dang, D. (2009). Msp430f543xa demo - usci_b0 i2c master tx multiple bytes to msp430 slave. Retrieved December, 2009, from <http://www.ti.com/lit/zip/slac357d>. Built with CCE Version: 3.2.2 and IAR Embedded Workbench Version: 4.11B.
81. Thanigai, P., & Morales, M. (2009). Msp430f543xa demo - usci_b0 i2c slave rx multiple bytes from msp430 master. Retrieved June, 2009, from <http://www.ti.com/lit/zip/slac357d>. Built with CCE Version: 3.2.2 and IAR Embedded Workbench Version: 4.11B.
82. Baker, B. (2011). *Designing with temperature sensors, part one: Sensor types*. Austin: Texas Instruments (EDN).
83. Baker, B. (2011). *Designing with temperature sensors, part two: Thermistors*. Austin: Texas Instruments (EDN).
84. Baker, B. (2011). *Designing with temperature sensors, part three: Rtds*. Austin: Texas Instruments (EDN).
85. Baker, B. (2011). *Designing with temperature sensors, part four: Thermocouples*. Austin: Texas Instruments (EDN).
86. Baker, B. (2012). *Designing with temperature sensors, part five: Ic-temperature sensors*. Austin: Texas Instruments (EDN).
87. Inc. Agilent Technologies. Application note 290: Practical temperature measurements. Technical report, Agilent Technologies, Inc., January 2012.
88. Bonnie, B. (2005). *A Baker's dozen: Real analog solutions for digital designers*. Boston, MA: Newnes.
89. Mancini, R. (Ed.). (2003). *Op Amps for everyone*. Boston, MA: Newnes.
90. Moghimi, R. (2000). Curing comparator instability with hysteresis. *Analog/AnalogDialogue*.
91. Inc. Maxim Integrated. Ds1843:fast sample-and-hold circuit. Technical report, Maxim Semiconductor Inc, Maxim Integrated Products, 120 San Gabriel Drive, Sunnyvale, CA 94086, 2012.
92. Inc. Maxim Integrated. Max5167: 32-channel sample/hold amplifier with output clamping diodes. Technical report, Maxim Integrated Products, Inc., Maxim Integrated Products, 120 San Gabriel Drive, Sunnyvale, CA, 2000.
93. Vega, C. A. (2005). A switched opamp comparator to improve the conversion rate of low-power low-voltage successive approximation adcs. Master's thesis, U. of Puerto Rico at Mayaguez, 2005.
94. Hauser, M. W. (1991). Principles of oversampling/d conversion. *Journal of Radio Engineering Society*, 39(1/2), 3-26.

Author Biography

Manuel Jiménez-Cedeño Professor Jiménez has over twenty-five years of experience teaching and developing embedded systems applications using microprocessors. After earning his Ph.D. degree in Electrical Engineering from Michigan State University in 1999, he joined the University of Puerto Rico at Mayaguez where he currently holds a position as professor in the Electrical and Computer Engineering Department. His current research and teaching interests include embedded systems design, rapid systems prototyping, and automated characterization of electronic devices and systems. His work has produced over one hundred refereed articles published in conferences and journals around the world. Professor Jiménez is a professional member of the IEEE and ASEE.

Rogelio Palomera-García Obtained his degree as Docteur es Sciences from the Swiss Federal Institute of Technology at Lausanne, Switzerland in 1979. After six years in a research and graduate teaching position at the Research and Higher Education Center at Ensenada (CICESE) in Mexico, he moved to the University of Puerto Rico at Mayaguez where he has been teaching since 1985. His work and teaching interests include analog, digital, and embedded electronics among other topics. He is a member of the IEEE and the Japan Institute of Information, Electronic and Communication Engineers (IECE).

Isidoro Couvertier-Reyes Obtained his degree of Doctor of Philosophy in Electrical Engineering from the Louisiana State University-Baton Rouge in 1996. From that moment he has been teaching at the University of Puerto Rico at Mayaguez. His work and teaching interests include Engineering Education, Computer Networking, Microprocessors, Computer Arithmetic, Computer Architecture, and Digital Logic. His MS degree is from the University of Wisconsin-Madison and his Baccalaureate from the University of Puerto Rico at Mayaguez. He is a senior member of the IEEE.

Index

Symbols

Default
 definition, 230

Break
 definition, 229

Switch-structure, 230

While-structure, 231

#define, 224

do-while-structure, 231

if-structure, 229

A

Absolute maximum ratings, 250

Actual parameter, 226

AD converter (ADC), 66

Address, 97

Address Bus, 108

Addressing Mode, 88, 132, 170

- direct, 133
- immediate, 133
- implicit, 132
- indexed, 133
- indirect, 133
- indirect register, 133
- register, 133
- relative, 134

Addressing mode

- base indexed, 133

Addressing Modes, 86, 119, 159

- absolute, 203
- direct, 203
- register, 173

Aliasing, 556

- antialiasing filter, 556

Allocation, 200

Alphanumeric codes, 71

- ASCII code, 71

ALU, 88, 96

Analog-to-Digital Converter (ADC), 110

Analog-to-digital converter (ADC), 532, 557, 572

- delta-sigma ADC, 565
- differential input, 557
- integrating ADC, 572
- slope ADC, 572
- successive approximation, 573

Antialiasing, 545

Arithmetic Operations, 89

Assembler, 108, 124, 158

- program location counter (PLC), 168, 200
- section location counter, 200
- section program counter, 200
- two pass, 160

Assembly Language, 122

Assembly language, 158

Assembly Time Constants, 161

B

B, 219

Background Mode Emulator, 16

Basic CPU interface, 249

BCPL, 219

Bias, 50

Big Endian Convention, 101

Binary coded decimal

- representation, 51

Binary coded decimal (BCD)

- 8421BCD, 51

- 84-2-1BCD, 51
 - arithmetic., 52
 - BCD addition, rule for, 52
 - BCD subtraction, rule for, 53
 - packed BCD, 51
 - unpacked BCD, 51
 - Binary system
 - arithmetic operations
 - addition, carry, 41
 - addition, complete, 41
 - division , 46
 - multiplication , 46
 - rules of subtraction, 42
 - subtraction , 42
 - subtraction, borrow, 42
 - complete subtraction, 42
 - rules of addition, 41
 - Bit, 31
 - fixed number of bits, 49
 - least significant bit, 32
 - most significant bit, 32
 - Bitwise operations, 89
 - Boot loader, 287
 - Boot Sequence, 287
 - Boot sequence operation, 288
 - Branch, 127
 - Brownout, 284
 - Buffered Input, 116
 - Bug, 143, 157
 - Bus
 - address, 97
 - address word, 97
 - control, 98
 - Data Bus, 105
 - data width, 97
 - definition, 96
 - Bus Interface Logic, 89
 - By-reference, 232
 - definition, 232
 - By-value, 232
 - definition, 232
 - Bypassing, 257
 - capacitor, 257
 - Byte, 31
 - integer representation, 49
 - Byte Instructions, 173
- C**
- C Preprocessing Directives
 - #define, 224
 - #include, 224
 - #pragma, 224
 - Calculated Branching, 320
 - Capacitive Touch Sensor, 384
 - Carry Flag, 185
 - Central Processing Unit, 81
 - Character constants, 71
 - CISC architecture, 85
 - Clock frequency
 - selection criteria, 264
 - Clock parameters, 260
 - drift, 263
 - duty cycle, 262
 - frequency, 261
 - jitter, 263
 - stability, 262
 - swing, 261
 - Clock sources
 - crystal oscillators, 266
 - external, 265
 - internal, 266
 - RC oscillators, 266
 - Comments, 124, 160
 - Communication protocol, 472
 - Comparator, 544, 546
 - hysteresis, 547
 - hysteresis comparator, 547
 - MSP430 comparator A, 550
 - MSP430 comparator A+, 550
 - MSP430 comparator B, 550
 - Compiler, 157
 - cross compiler, 169
 - definition, 220
 - Conditional Jumps, 190
 - Conditioning, 532
 - Configurable Device, 155
 - Constant Generator, 174
 - Constants
 - compiling time, 225
 - Continuous representation, 66
 - Control Path, 86
 - Control Structures, 190
 - IF-ELSE, 191
 - IF-THEN, 191
 - Control Unit, 87
 - Core Instructions, 138
 - COTS, 17
 - CPU, 81, 86
 - active mode, 207, 326
 - exiting sleep mode, 326
 - low-power mode, 326
 - read, 96
 - reset, 88
 - write, 96
 - Cross-compiler, 220

- Crystal oscillator, 266
 - Colpitts, 268
 - equivalent circuit, 267
 - Pierce, 268
 - startup time, 269
 - topologies, 267

- D**
- DaisyChain, 307
- Data
 - memory initialized constants, 202
- Data Address, 101
- Data Bus, 108
- Datapath, 86
- Debugger, 39, 157
- Debugging, 143, 157
- Decimal arithmetic, 51
- Decoupling, 258
- Delay Loop, 127, 190
- Delta Sigma ADC ($\Delta\Sigma$ ADC), 576
- Dennis Ritchie, 219
- Destination, 89, 120
- Detector, 533
- Differential connection, 470
- Digit, 33
 - least significant digit, 33
 - most significant digit, 33
 - weight of, 33
- Digital to analog converter (DAC), 553, 565
 - bipolar DAC, 566
 - parallel DAC, 566
 - serial DAC, 566
- Digital-to-Analog Converter (DAC), 110
- Digital-to-analog converter (DAC), 532
 - charge distribution, 567
- Directive, 160
 - #define, 161
 - C-type preprocessor directive, 160, 161
 - CODE, 201
 - CSTACK, 201
 - DATA, 201
 - data definition or data allocation, 203
 - DC16, 167
 - DW, 167
 - END, 167
 - ENDM, 199
 - LOCAL, 199
 - MACRO, 199
 - ORG, 161, 167, 200
 - RESET, 201
 - RSEG, 201
 - segment directives, 200
 - SET, 161
- Directives, 123, 160
 - #include, 161
 - EQU, 161
- Disassembler, 158
- Display Types, 413
 - alphanumeric, 414
 - discrete, 413
 - graphic, 414
 - numeric, 414
- Documentation, 200
- Double Word Address, 101
- Double Word Boundary, 102
- Dropout voltage, 254
- Dynamic memory allocation
 - definition, 219

- E**
- Electromigration, 22
- Embedded Software
 - kernel, 10
 - services, 11
 - system resources, 10
 - tasks, 10
- Embedded System
 - computer-on-a-chip, 83
 - definition, 1
 - lifecycle, 13
 - structure, 7
- Embedded Systems
 - cost, 15, 17
 - energy, 15
 - firmware, 9
 - functionality, 15
 - maintainability, 15
 - performance, 19
 - performance, 15
 - size, 15
 - time to market, 15
- Emulated Instructions, 138
- Emulator, 150
- Evaluation Kit, 150
- Excess, 50
- Executable Code, 157
- Executable File, 124

- F**
- Fetching, 142
- Filter
 - active filter, 545
 - low-pass, 545

Firmware, 7, 9
 Fixed-point representation, 59
 extra scaling, 61
 format Fm.n, 59
 step, 59
 Flag, 91, 116
 carry, 186
 Global interrupt enable (GIE), 213
 Flag Register, 91
 Flags
 carry, 91
 general interrupt enable, 91
 interrupt, 91
 N, 178
 negative, 91
 overflow, 91
 V, 178
 zero, 91
 flags, 125
 Flash Memory
 controller, 360
 erase memory, 358
 programming mode, 358
 read mode, 358
 wear leveling, 360
 Floating Gate MOS Transistor, 359
 Floating point numbers, 62
 Floating point representation, 62
 denormalized mantissa, 62
 double precision, 62
 mantissa, 62
 normal position, 62
 normalized mantissa, 62
 single precision, 62
 Framing error, 474, 475
 Frequency-locked loop, 271
 Full scale range, 558
 Function
 definition, 232
 parameters, 210
 Functions, 207

G

GPIO, 377
 configuration, 380
 Electrical Characteristics, 385
 pin oscillator, 384
 structure, 378
 GPIO:low and high voltage rules, 386
 GPIO:Nominal Voltages, 385

H

H-bridge, 448
 Hardware Emulation, 16
 Hardware Model, 85
 Hardware Multiplier, 89, 96, 190, 195
 Hardwired Instructions, 170
 Harvard Architecture, 104
 Header File, 161
 Header Files, 161, 162
 Hex Machine Language, 157
 Hex notation, 71
 Hexadecimal system
 arithmetic operations, 46
 High Level Language, 122
 High-level Language
 C language, 219
 Hysteresis value, 547

I

I/O Port, 8
 parallel, 115
 serial, 115
 I/O Ports, 377
 I/O Subsystem, 109
 I/O-mapped I/O, 110
 standard IO, 110
 I/O subsystem, 82
 I2C
 bus arbitration, 499
 clock synchronization, 498
 data transfer, 497
 interface structure, 496
 message, 498
 multimaster configuration, 498
 NACK, 498
 start, 497
 stop, 497
 wait state, 499
 IAR Directives
 END, 200
 IDE
 definition, 221
 Inlining, 238
 Input
 device or peripheral, 109
 transaction, 109
 Instruction
 addressing mode, 122
 decode, 88
 execution, 88
 fetch, 88, 90
 instruction cycle, 87

- issuing, 90
- machine language, 122
- mnemonic, 124, 158
- OpCode, 122
- operand, 122
- operands, 158
 - destination, 159
 - source, 159
- queue, 90
- register, 90
- Instruction Address, 101
- Instruction Cycle, 87, 189
- Instruction Pointer, 90
- Instruction Register, 88
- Instruction Set, 86, 119
- Instruction Word, 122
- Instruction word, 158
- Instructions, 123, 165, 206
 - arithmetic, 178
 - arithmetic-logic, 125
 - bit, 184
 - call, 164, 208
 - clear bit AND, 183
 - clear bit BIC, 183
 - core, 170
 - data transfer, 124, 176
 - decimal addition, 178
 - decimal addition dadd, 178
 - dint, 189
 - eint, 189
 - emulated, 170
 - jump, 90
 - logic instructions, 181
 - nop, 189
 - pop, 91
 - program control, 126
 - pull, 91
 - push, 91
 - register control, 181, 185
 - ret, 208
 - reti, 159, 215
 - set bit BIS, 183
 - sign extension sxt, 178
 - swap, 124
 - swap bytes, 177
 - test bit BIT, 183
 - xor, 183
- Instrumentation
 - scaling, 588
- Integer, 33, 48
 - BCD representation, see Binary Coded Decimal (BCD)51
 - unsigned binary representation, 49
- Integers
 - biased or excess notation, 50
- Interfacing Switches, 396
- Interpreter, 124, 157
- Interrupt
 - interrupt controller, 308
 - priority arbiter, 308
- Interrupt, 91, 116, 131, 140
 - acknowledgment cycle, 306, 308
 - acknowledgment signal, 306
 - auto-vector, 306
 - auto-vectored, 304
 - daisy chain arbitration, 307, 310
 - daisy-chain, 307
 - dummy ISR, 322
 - false trigger, 320, 322
 - fixed address, 305, 306
 - fixed vector, 306
 - full-vectored, 306
 - GIE, 312
 - in the MSP430, 310
 - interrupt controller, 307
 - interrupt driven, 116, 301
 - interrupt driven programming, 207
 - interrupt request, 139, 207
 - Interrupt return, 142
 - interrupt service routine (ISR), 131
 - interrupt vector, 204, 240, 306
 - interrupt vector table, 108, 204
 - ISR, 140
 - maskable, 141
 - multi-source, 318
 - nesting, 323
 - NMI, 141
 - non-maskable, 311
 - non-maskable , 301
 - non-vectored, 304, 305
 - prioritization, 324
 - priority service, 305
 - reentrancy, 324
 - reset vector, 88
 - service process, 303
 - simultaneous requests, 307
 - software interrupt, 301
 - software interrupts, 207
 - vector interrupt, 164
 - vector table, 306, 310
 - vectored, 304, 306
- interrupt
 - latency, 323
 - latency reduction, 323
- Interrupt Acknowledgement, 301
- Interrupt Controller, 308

Interrupt Service Routine, 140, 164, 165
 Interrupt Service Routine (ISR), 207, 214
 Interrupt:edge sensitive, 300
 Interrupt:level sensitive, 300
 Interrupt:maskable, 301
 Interrupts
 MSP430 maskable, 312
 Interrupt:identification mechanisms, 304
 Intrinsic Functions, 238
 IO Ports
 direction register PxDIR, 114
 function select register PxSEL, 115
 input register PxIN, 115
 MSP430 IO port architecture, 114
 output register PxOUT, 115
 IO subsystem
 configuration register, 112
 control registers, 112
 data registers, 112
 mode register, 112
 status registers, 112
 IP address, 80
 IrDA, 518
 Iteration Loop, 190
 Iteration Loops, 127

J

Jittering, 565
 JTAG, 146
 debugger, 16
 definition, 16
 JTAG interface, 287
 Jump, 93
 conditional, 126
 unconditional, 126
 Jump Table, 320
 Jump table, 288

K

Keypad
 Interface, 410
 Scanning Algorithm, 412
 Structure, 409
 Keypad Interface
 dedicated ICs, 412

L

Label, 128, 134, 160
 Latched Input, 116
 LED Interface, 414
 BJT-based, 440

 Darlington, 442
 direct, 416
 discrete, 415
 MOSFET, 439
 LED Interfacing
 7-segment, 416
 Library, 212
 Linker, 124, 200, 212
 Linux, 219
 List File, 160, 162
 Little Endian Convention, 101
 Load-store Architecture, 125
 Local Constants, 209
 Local Variables, 209
 Logic Operations, 89
 Loop Structure
 DO-UNTIL, 193
 REPEAT-UNTIL, 193
 WHILE, 193
 Loop Structures, 190
 Low-power Modes, 324

M

Machine code, 219
 Macro, 198
 expansion, 198
 mnemonic of., 198
 Macro Directives, 124
 Maintainability, 26
 Market Window, 24
 Mask, 181
 Maskable Interrupts, 91
 Memory, 98
 bank, 100
 data, 8, 103
 data memory, 103
 dynamic RAM (DRAM), 99
 even address bank, 101
 ferro electric RAM (FRAM), 99
 flash memory, 358
 global map, 108
 high bank, 101
 in-system programmable, 99
 low bank, 101
 map, 107
 memory address, 98
 memory word, 98
 NVRAM, 99
 odd address bank, 101
 partial map, 108
 program, 8
 program memory, 103

- RAM, 103, 105
- Random-access, 103
- read only memory (ROM), 99
- segment, 101
- static RAM (SRAM), 99
- write/read, 99
- Memory Map, 86
- Memory Segment, 200
- Memory-Mapped I/O, 111
- Memory-mapped I/O, 372
- Microcomputer
 - definition, 81
- Microcontroller, 81, 83
 - family, 84
- Microprocessor Unit, 86
- Mnemonic, 158
- Modular Programming, 207
- Motor Interface, 447
 - DC Motors, 447
 - H-bridge, 448
 - PWM, 448
 - Servo-motors, 449
 - Stepper Motors, 450
- MSP430
 - address-bit multiprocessor mode, 503
 - basic clock module, 274
 - basic clock module+, 276
 - break error, 503
 - enhanced USCI, 527
 - eUSCI_A, 527
 - eUSCI_B, 528
 - FLL clock module, 271
 - FLL+ clock module, 272
 - idle-time multiprocessor mode, 502
 - LCD Controller, 434
 - LCD_A Controller, 435
 - LCD_B Controller, 437
 - LCD_C Controller, 437
 - NMI, 291
 - non-maskable interrupt sources, 311
 - POR, 291
 - PUC, 291
 - reset structure, 290
 - serial interface, 501
 - supply voltage supervisor, 291
 - UART baud rates, 503
 - UART errors, 503
 - UART interrupts, 506
 - Unified clock system, 277
 - USART, 501
 - USART SPI mode, 508
 - USCI, 514
 - USI, 510
 - USI in I2C mode, 512, 521
 - USI in SPI mode, 510, 521
 - USI in UART mode, 515
- MSP430 ADC's
 - ADC10, 579
 - ADC12, 579
 - sigma-delta converters, 579
- MSP430 DAC
 - DAC_xCTL, 570
- MSP430 peripherals
 - supply voltage supervisor, 74
- MSP430 Port
 - Interruptible Port
 - interrupt edge select register (PxIES), 212
 - interrupt enable register (PxIE), 212
- MSP430:GPIOs, 382
- Multi-drop links, 472
- Multiply-And-Accumulate, 195

- N**
- Nibble, 31
- NMI, 301
- Non-maskable Interrupt, 141
- NRE Costs, 17
- Number Systems
 - Conversion between systems
 - Hex to Binary, 39
- Number systems
 - conversion by repeated multiplication, 36
 - base r conversion to decimal, 35
 - binary system, 34
 - binary to octal conversion, 38
 - conversion between systems
 - binary to hex, 39
 - conversion by repeated division, 36
 - conversion stopping criteria, 37
 - decimal fractional conversion to base r , 36
 - decimal integer conversion to base r , 36
 - decimal system, 34
 - fractional part, 33
 - hexadecimal system, 34
 - integer part, 33
 - octal system, 34
 - octal to binary conversion, 38
 - positional system, 33
 - fixed-point, 33
 - fixed-radix, 33
- Nyquist principle, 555
 - Nyquist frequency, 555
 - Nyquist rate, 555
- Nyquist rate converters, 576

O

Object Code, 212
 Object File, 160
 Opcode, 170
 Operand, 158
 destination, 159
 source, 159
 Operational amplifier (OA), 538
 buffer, 540
 closed-loop gain, 539
 difference amplifier, 541
 ideal model, 538
 infinite gain, 539
 instrumentation amplifier, 542
 inverting adder, 540
 inverting input, 538
 inverting amplifier, 540
 inverting integrator, 540
 linear region, 538
 non-inverting input, 538
 noninverting amplifier, 540
 open-loop gain, 538
 subtractor, 540
 unity gain amplifier, 540
 voltage follower, 540
 Operational amplifier(OA)
 current to voltage converter, 540
 Operational amplifiers
 Howland source, 543
 programmable gain amplifier (PGA), 545
 voltage-to-current converter, 543
 Operational amplifier (OA)
 differential amplifier, 540
 Optimizing Compiler
 definition, 221
 Orthogonal Architecture, 126
 Orthogonality, 137
 Output
 transactions, 109
 devices or peripherals, 109
 Overflow, 49, 57
 unsigned integers, 50
 Oversampling converters, 576
 oversample ratio (OSR), 577

P
 Parallel Interface, 115
 Parity
 even, 73
 odd, 73
 Parity bit, 72
 Parity check, 475

Parity error, 475
 Parity even, 475
 Parity odd, 475
 Peripherals, 82
 Physical Address, 100
 PnDIR, 166
 Pointer, 227
 address-of operator, 227
 at-address operator, 227
 definition, 219
 Polling, 116, 139, 140, 212
 Power on reset (POR), 75
 Power supply noise, 256
 Power-on reset circuit, 282
 Power-on-reset, 282
 Preprocessor, 224
 Princeton Architecture, 104
 Priority Arbiter, 308
 Processor Status Word, 91
 Processor's power source, 250
 Program, 103, 143
 Program Counter, 209
 Program Memory, 200
 Programmable, 155
 Programmable hardware, 74
 Programmer Model, 85
 Programming Language, 155
 assembly language, 155
 binary machine language, 157
 high level language, 85
 high-level language, 155
 machine language, 155
 Pull-down Resistor, 115
 Pull-up Resistor, 115

Q

Quantization, 66, 553, 555
 biased binary code, 559
 bottom offset, 563
 differential linearity error (DLE), 563
 differential non linearity (DNL), 563
 effective number of bits (ENOB), 563
 full scale offset error, 563
 gain error, 563
 input code width, 563
 integral linearity error (ILE), 563
 integral non linearity (INL), 563
 LSB resolution, 67, 558
 n-bit resolution, 66
 normal, 67
 offset binary code, 559
 offset error, 562

- offset quantization, 67
 - precision, 66, 558
 - quantization error, 66, 554, 557
 - quantization level, 557, 561
 - quantization levels, 66
 - resolution, 66, 558
 - signal-to-noise and distortion (SINAD), 563
 - signal-to-noise ratio (SNR), 563
 - signal-to-noise ratio SNR, 562
 - straight binary code, 559
 - straight binary encoding, 67
 - two's complement, 560
 - zero scale error, 562
- R**
- R-2R ladder configuration, 566
 - Radix
 - radix point, 33
 - Rail, 538
 - rail-to-rail, 538
 - Real-Time Operating System RTOS, 149
 - Recommended operating conditions, 250
 - Record
 - definition, 233
 - Recurrent Cost, 17
 - Reference current circuit, 543
 - Register
 - address, 89
 - PC, 122
 - program counter, 143
 - stack pointer (SP), 166
 - status register (SR), 121
 - register
 - PSW, 301
 - Register Transfer Notation (RTN), 120
 - Registers, 89
 - accumulator, 125
 - constant generator, 95
 - data, 89
 - general purpose, 89
 - instruction, 90
 - instruction register (IR), 104
 - PC, 189, 208
 - program counter, 90
 - program counter (PC), 88, 95, 104
 - stack pointer (SP), 90, 95, 125
 - status, 91
 - status register, 125, 207, 327
 - status register (SR), 95
 - registers
 - specialized, 90
 - Relocatable Assembly Code, 200
 - relocatable section, 200
 - relocatable segment directives, 200
 - Relocatable Section
 - offset address, 200
 - Reset, 142, 282
 - address, 283
 - brown out reset (BOR), 143
 - POR circuit, 284
 - power on reset (POR), 143
 - requirements, 282
 - signal effect, 282
 - timing, 282
 - Reset address, 288
 - auto-vector, 288
 - fixed vector, 288, 289
 - jump table, 288
 - Reset Vector, 88, 164, 168, 204
 - Reset vector, 289
 - Reset Vector Allocation, 167
 - Resource Management, 10
 - RISC Architecture, 126
 - RISC architecture, 85
 - RMS value, 562
 - ROM Monitor, 16
 - Rounding [A], 559
 - RS-485, 487
 - termination impedance, 487
 - RTOS, 10
- S**
- Sample and hold circuit, 564
 - hold phase, 564
 - sampling phase, 564
 - Sampling, 555
 - Nyquist principle, *see* Nquist principle555
 - sampling frequency, 555
 - sampling period, 555
 - sampling rate, 555
 - Schmitt-trigger Input, 390
 - Schmitt-trigger Input, GPIO
 - Schmitt-trigger, 379
 - Scientific notation, 62
 - sec:ExecutableCode, 165
 - sec:ExecutableCodeII, 206
 - Sensor
 - active, 533
 - IC sensors, 537
 - passive, 533
 - RTD, 537
 - Seebeck voltage, 537
 - thermistors, 537

- thermocouples, 537
- Sensors, 531
 - accuracy, 535
 - analog, 533
 - dynamic linearity, 535
 - dynamic range, 534
 - full scale, 534
 - hysteresis, 535
 - linearity, 535
 - non linearity, 535
 - offset, 535
 - photovoltaic mode, 541
 - precision, 534
 - range, 534
 - resolution, 535
 - response time, 535
 - sensitivity, 534
- Separate memory and I/O spaces, 371
- Serial
 - transceiver, 471
- Serial communication
 - asynchronous, 473
 - baud rate, 469
 - baud rate clock, 479
 - baud rate error, 481
 - bit rate, 469
 - bps, 469
 - data packet, 473
 - differential channel, 486
 - framing error, 475, 477
 - full-duplex, 472
 - half duplex, 471
 - I2C, 495
 - master/slave, 492
 - MSP430 USCI, 514
 - MSP430 USI, 510
 - NULL-modem cable, 484
 - overrun error, 478
 - packet footer, 473
 - packet format, 474
 - packet header, 473
 - parity bit, 475
 - parity error, 475
 - physical standards, 483
 - point-to-point, 472
 - RS-232, 483
 - RS-422, 486
 - RS-485, 487
 - simplex, 471
 - simplex multidrop, 487
 - SPI, 493
 - synchronous, 492
 - UART, 476, 501
 - USART, 476, 501
 - USB, 488
 - voltage requirement, 480
- Serial communications
 - auto-baud, 517
 - break character, 517
- Serial vs. parallel communication, 469
- Shift Instructions, 185
- Sigma Delta ADC ($\Sigma\Delta$ ADC), 576
- Sign Bit, 54
- Sign bit, 63
- Sign extension, 55
- Signal chain, 531
 - analog, 531
- Signed numbers, 48
- Single-ended connection, 470, 486
- Sixteen's complement, 47
- Smoothing, 554
- Smoothing filters, 570
- Source, 89, 120
- Source Code, 156, 160
 - main code, 165
- Source File, 123, 159, 162, 167, 200
 - absolute, 167, 200
 - executable code, 165, 206
 - relocatable, 200
 - source statement, 159
- Special Function Registers (SFR), 117
- Stack, 90, 128, 211
 - last-in-last-out (LIFO), 128
 - pop operation, 128
 - pull operation, 128
 - push operation, 128
 - top of stack (TOS), 128
- Status Register
 - GIE flag, 301
- Stepper Motors, 450
 - Bifilar Winding, 460
 - Bipolar Winding, 458
 - Full Step, 451
 - Half Step, 451
 - Hybrid, 460
 - Integrated Drivers, 464
 - Microstepping, 461
 - Permanent Magnet, 454
 - Single Phase, 451
 - Software Plan, 463
 - Unipolar Winding, 454
 - Variable Reluctance, 450
 - Wave Mode, 451
- Storage duration
 - definition, 226
- Storage Permanence, 99, 356

- nonvolatile, 99
 - volatile, 99
 - Stored-Program Computer, 81
 - String, 71
 - Strongly-typed
 - definition, 219
 - Subroutine, 126, 131, 164, 207
 - passing data, 209
 - reusable, 207
 - subroutines, 165
 - Switch
 - arrays, 408
 - linear, 408
 - matrix, 409
 - bounce time, 400
 - contact bounce, 399
 - debounce, 401
 - counting, 405
 - hardware, 401
 - ICs, 403
 - polling, 405
 - RC, 401
 - software, 404
 - SR latch, 401
 - interface, 397
 - types, 397
 - Switches
 - DIP, 408
 - Symbolic Names, 160
 - System Bus, 81, 82
 - address bus, 96
 - control bus, 96
 - data bus, 96
 - transaction, 97
 - System Buses, 117
 - System clock, 260
 - System Memory, 82
- T**
- Task Communication, 11
 - Task Management, 10
 - Ten's complement, 44
 - Testing, 143
 - Threshold detection, 546
 - Time-to-market, 24
 - Timer, 110, 330
 - comparator, 331
 - compare register, 331
 - counter, 331
 - divider, 331
 - event counter, 332
 - interrupt signal, 331
 - interval timer, 332
 - output compare signal, 332
 - overflow signal, 331
 - Timing Diagrams, 117
 - Total Harmonic Distortion (THD), 563
 - Transducer, 554
 - Transductor, 532, 533
 - Two Pass Assembler, 134
 - Two's complement, 44
 - computation, 45
 - Two-pass Assembler, 124
 - Twos Complement Addition, 93
- U**
- UART, 476
 - Unconditional Jump, 190
 - Underflow, 65, 92
 - Unicode, 73
 - Universal Serial Bus, 485, 488
 - Unix, 219
 - Unregulated source, 252
 - Unsigned numbers, 48
 - USB, 485
 - architecture, 489
 - cable, 490
 - devices, 489
 - host, 489
 - host controller, 489
 - Hub, 490
 - tier, 490
- V**
- Vacant Space, 109
 - Variable
 - data type, 227
 - definition, 225
 - scope of visibility, 226
 - Variable attributes
 - definition, 226
 - Variable Cost, 17
 - Variables, 203
 - initialized variable, 202
 - Vector table, 290
 - Von Neumann Architecture, 104
- W**
- Watchdog Timer, 110, 143, 166
 - Watchdog Timer (WDT), 336
 - Word, 31
 - byte, 31
 - doubleword, 31

- integer representation, [49](#)
 - integer representation with double word, [49](#)
 - nibble, [31](#)
 - quad, [31](#)
 - Word Address, [101](#)
 - Word Boundary, [102](#)
 - Word Instructions, [173](#)
 - Write Ability, [99](#), [356](#)
- Z**
- Zero Flag, [185](#)