Microprocessor and Microcontrollers

Part-5

educlash.com
Just another Way To Learn

NOP	Do nothing and go to the next instruction; NOP (no operation) is used to waste time in a software timing loop; or to leave room in a program for later additions; no flags are affected

5. CALLS AND SUBROUTINES

The method of changing program execution is using "interrupt" signals on certain external pins or internal registers to automatically cause a branch to a smaller program that deals with the specific situation. When the event that caused the interruption has been dealt with, the program resumes at the point in the program where the interruption took place. Interrupt action can also be generated using software instructions named calls.

Call instructions may be included explicitly in the program as mnemonics or implicitly included using hardware interrupts. In both cases, the call is used to execute a smaller, stand alone program, which is termed a routine or, more often, a subroutine.

1. Subroutines

A subroutine is a program that may be used many times in the execution of a larger program. The subroutine could be written into the body of main program everywhere it is needed, resulting in the fastest possible code execution. Using a subroutine in this manner has several serious drawbacks.

Common practice when writing a large program is to divide the total task among many programmers in order to speed completion. The entire program can be broken into smaller parts and each programmer given a part to write and debug. The main program can then call each of the parts, or subroutines, that have been developed and tested by each individual of the team.

Even if the program is written by one individual, it is more efficient to write an oft-used routine once and then call it many times as needed. Also, when writing a program, the programmer does the main part first. Calls to subroutines, which will be written later, enable the larger task to be defined before the programmer becomes bogged down in the details of the application.

2. Calls and the Stack

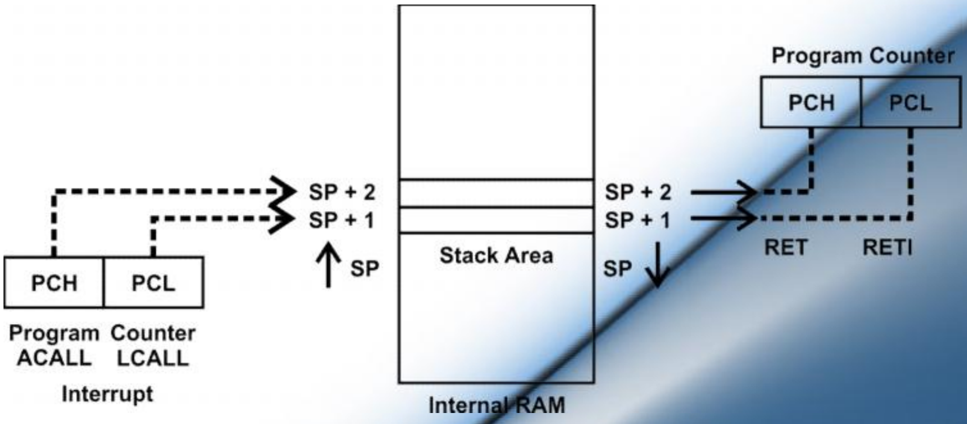
A call, whether hardware or software initiated, causes a jump to the address where the called subroutine is located. At the end of the subroutine the program resumes operation at the opcodes address immediately following the call. As calls can be located anywhere in the program address space and used many times, there must be an automatic means of storing the address of the instruction following the call so that program execution can continue after the subroutine has executed.

The stack area of internal RAM is used to automatically store the address, called the return address, of the instruction found immediately after the call. The stack pointer register holds the address of the last space used on the stack. It stores the return address above this place, adjusting itself upward as return address is stored. The term “stack” and “stack pointer” are often used interchangeably to designate the top of the stack area in RAM that is pointed to by the stack pointer.

Figure 2 diagram the following sequence of events:

1. A call opcodes occurs in the program software, or an interrupt is generated in the hardware circuitry.
2. The return address of the next instruction after the call instruction or interrupt is found in the program counter.
3. The return address bytes are pushed on the stack, low byte first.
4. The stack pointer is incremented for each push on the stack.
5. The subroutine address is placed in the program counter.
6. The subroutine address is executed.
7. A RET (return) opcode is encountered at the end of the subroutine.

FIGURE 2 Storing and Retrieving the Return Address.



8. Two pop operations restore the return address to the PC from the stack area in internal RAM.
9. The stack pointer is decremented for each address byte pop.

All of these steps are automatically handled by the 8051 hardware. It is the responsibility of the programmer to ensure that the subroutine ends in a RET instruction and that the stack does not grow up into data areas that are used by the program.

15.5.3 Calls and Returns

Calls use short or long range addressing: returns have no addressing mode specified but are always long range. The following table shows examples of call opcodes:

Mnemonic	Operation
ACALL sadd	Call the subroutine located on the same page as the address of the opcodes immediately following the ACALL instruction; push the address of the instruction immediately after the call on the stack
LCALL ladd	Call the subroutine located anywhere in program memory space; push the address of the instruction immediately following the call on the stack
RET	Pop two bytes from the stack into the program counter

Note that no flags are affected unless the stack pointer has been allowed to erroneously reach the address of the PSW special function register.

15.6 INTERRUPTS AND RETURNS

An interrupt is a hardware-generated call. Just as a call opcodes can be located within a program to automatically access a subroutine, certain pins on the 8051 can cause a call when external electrical signals on them go to a low state. Internal operations of the timers and the serial port can also cause an interrupt call to take place.

The subroutines called by an interrupt are located at fixed hardware addresses. The following table shows the interrupt subroutine addresses.

INTERRUPT	ADDRESS (HEX) CALLED
IE0	0003
TF0	000B
IE1	0013
TF1	001B
SERIAL	0023

When an interrupt call takes place, hardware interrupt disable flip-flops are set to prevent another interrupt of the same priority level from taking place until an interrupt return instruction has been executed in the interrupt subroutine. The action of the interrupt routine is shown in table below.

Mnemonic	Operation
RETI	Pop two bytes from the stack into the program counter and reset the interrupt enable flip-flops

Note that the only difference between the RET and RETI instructions is the enabling of the interrupt logic when RETI is used. RET is used at the ends of subroutines called by an opcodes. RETI is used by subroutines called by an interrupt.

The following program examples use a call to a subroutine.

```
ADDRESS   MNEMONIC              COMMENT
MAIN:     MOV 81h, #30h         ; set the stack pointer to
                                30h in RAM
          LCALL SUB             ; push address of NOP;
                                PC = #SUB; SP = 32h
          NOP                   ;return from SUB to this
                                opcodes
          . . .
          . . .
SUB       MOV A, #45h           ;SU loads A with
                                45h and returns
          RET                   ;pop return address to PC;
                                SP = 30h
```

In the following example of an interrupt call to a routine, timer 0 is used in mode 0 to overflow and set the timer 0 interrupt flag. When the interrupt is generated, the program vector to the interrupt routine, resets the timer 0 interrupt flag, stop the timer, and returns.

```
ADDRESS   MNEMONIC              COMMENT
          .0RG 0000h            ;begin program at 0000
          AJMP OVER             ;jump over interrupt
                                subroutine
          .0RG 000Bh            ;put timer 0 interrupt
                                subroutine here
          CLR 8Ch               ;stop timer 0; set TRO = 0
          RETI                  ;return and enable
                                interrupt structure

OVER:                           ; enable the timer 0
          MOV 0A8h, #82h        interrupt in the IE
                                ;set timer operation, mode
          MOV 89h, #00h         0
                                ;clear TL0
          MOV 8Ah, #00h

          MOV 8Ch, #00h         ;clearTH0
          SET 8Ch               ;start timer 0; set TR0 = 1
```

;
;
;
; the program will continue on and be interrupted when the timer has timed out

15.7 SUMMARY

- □ Jump alter program flow by replacing the PC counter contents with the address of the jump address.
- □ Jumps have the following ranges:
  Relative : up to PC +127 bytes, PC – 128 bytes away from PC
  Absolute short : anywhere on a 2K-byte page
  Absolute long : anywhere in program memory
- □ Jump opcodes can test an individual bit, or a byte, to check for conditions that make the program jump to a new program address.
- □ Bit jumps all operate according to the status of the carry flag in the PSW or the status of any bit-addressable location.
- □ Unconditional jumps do not test any bit or byte to determine whether the jump should be taken. The jump is always taken. All jump ranges are found in this group of jumps, and these are the only jumps that can jump to any location in memory.

15.8 REVIEW QUESTIONS

1. Explain the relative range.
2. What do you mean by absolute short range and long range?
3. Explain bit and byte jump instructions.
4. Explain subroutine program.
5. Explain different return instructions.

15.9 REFERENCE

- ➢ The 8051 Microcontroller by Kenneth J. Ayala, Publisher: Thomson Delmar Learning
- ➢ The 8051 Microcontroller And Embedded Systems Using Assembly And C, 2/E By Mazidi and Mazidi, Publisher: Pearson Education India

❖❖❖❖

16

8051 PROGRAMS

1. To search a number from a given set of numbers. The end of the data is indicated by 00.

Memory Location	Label	Instruction	Comment
0000		MOV R1,#30H	Starting location of the list
0002	L1	MOV A,@R1	Number copied into accumlater
0003		CJNE A,#00,L3	Compared for the end of a list with 00H
0006	L3	CJNE A,#0AH,L2	Compared with No. 0AH
0009		MOV A,R1	Moving the content of R1 to A
000A		MOV R2,A	Store the number
000B	HERE	SJMP HERE	End of the program
000D	L2	INC R1	Get the next number
000E		JMP L1	Jump to L1

2. Finding the average of signed numbers.

Memory Location	Label	Instruction	Comment
0000		MOV R0,#30H	Starting location of a list
0002		MOV R2,#00H	Initialize R2 to store carry
0004		MOV R1,#05H	Counter is set to 05.
0006		MOV B,R1	Moving the content of R1 to B
0008		MOV A,#00H	Clear the accumulator

000A	L2	ADD A,@R0	Adding the value to accumulator
000B		JB OV, HERE	Check overflow flag
000E		JNC L1	If there is no carry jump to L1
0010		INC R2	If there is carry, increment R2
0011	L1	DJNZ R1, L2	Decrement R1, if it is not equal to 0, jump L2
0013		DIV AB	Divide accumulator with B
0014		MOV 40H,R2	Copying R2 to memory address 40H
0016		MOV 41H,A	Copying A to memory address 41H
0018		MOV 42H,B	Copying B to memory address 42H
001B	HERE	SJMP HERE	End of the program

2.A. Average of string of numbers.

Memory Location	Label	Instruction	Comment
0000		MOV R0,#30H	Starting location of a list
0002		MOV R2,#00H	Initialize R2 to store carry
0004		MOV R1,#05H	Counter is set to 05.
0006		MOV B,R1	Moving the content of R1 to B
0008		MOV A,#00H	Clear the accumulator
000A	L2	ADD A,@R0	Adding the value to accumulator
000B		JNC L1	If there is no carry jump to L1
000D		INC R2	If there is carry, increment R2
000E	L1	DJNZ R1,L2	Decrement R1, if it is not equal to 0, jump L2
0010		DIV AB	Divide accumulator with B

0011		MOV 40H,R2	Copying R2 to memory address 40H
0013		MOV 41H,A	Copying A to memory address 41H
0015		MOV 42H,B	Copying B to memory address 42H
0018	HERE	SJMP HERE	End of the program

3. Multiplication of signed numbers.

Memory Location	Label	Instruction	Comment
0000		MOV R1,#01H	Store 01H in reg. R1
0002		MOV R0,#30H	Initialize memory location
0004		MOV A,@R0	Number copied into accumlater
0005		MOV R7,A	Number copied into reg. R7
0006		RLC A	Rotate accumulator to check the carry
0007		JNC L1	If there is no carry, jump to L1
0009		MOV A,@R0	Number copied into accumlater
000A		CPL A	Taking 1st complement
000B		INC A	Taking 2‟s complement
000C		INC R1	Increment reg. R1
000D	L1	MOV A,R7	Number copied into reg. R7
000E		INC R0	Getting the next number
000F		MOV A,@R0	Number copied into accumlater
0010		MOV B,A	Number copied into reg. B
0012		RLC A	Rotate accumulator to check the carry
0013		JNC L2	If there is no carry, jump to L2

0015		CPL A	Taking 1st complement
0016		INC A	Taking 2‟s complement
0017		MOV B,A	Number copied into reg.B
0019		DEC R1	Decrement the content of R1
001A	L2	MOV A,R7	Number copied from reg. R7 to A
001B		MUL AB	Multiplying content A & B
001C		INC R0	Increment memory location R0
001D		MOV @R0,A	Copying A to memory
001E		INC R0	Increment memory location R0
001F		MOV @R0,B	Copying B to memory
0021		INC R0	Increment memory location R0
0022		MOV A,R1	Number copied from reg. R1 to A
0023		CJNE A,#02H,L3	If A is not equal to 02H, jump to L3
0026		MOV @R0,#01H	If A is equal to 02H, store 01 at memory
0028	HERE	SJMP HERE	Short jump
002A	L3	MOV @R0,#00H	Store 00 at memory
002C		JMP HERE	Jump to HERE

4. Convert the BCD 0111 0101 number to two binary numbers and transfer this number to registers.

Memory Location	Label	Instruction	Comment
0000		MOV A,#75H	Storing BCD no. in accumulator
0002		MOV B,A	Copying the number in reg. B
0004		ANL A,#0F0H	Masking lower nibble

0006		SWAP A	A_{3-0} swap A_{7-4}
0007		MOV 19H,A	Store the number at 19H
0009		MOV A,B	Copying original number in accumulator
000B		ANL A,#0FH	Masking higher nibble
000D		MOV 18H,A	Store the number at 18H
000F	HERE	JMP HERE	End of the program

5. To find y where y = x2 + 2x + 5 and x is between 0 and 9.

Memory Location	Label	Instruction	Comment
0000		MOV R2,#0AH	Store 0AH in reg. R2
0002		MOV R0,#60H	Store 60H in reg. R2
0004		MOV R1,#70H	Store 70H in reg. R2
0006		MOV R3,#00H	Initialize R3 with 00H
0008	L1	MOV A,R3	Copying R3 to Accumulator
0009		MOV @R0,A	Copying A to memory address 60H
000A		INC R3	Increment R3
000B		INC R0	Increment R0
000C		DJNZ R2,L1	If R2≠0AH, Jump to L1
000E		MOV R2,#0AH	Store 0AH in reg. R2
0010		MOV R0,#60H	Store 60H in reg. R2
0012		MOV R1,#70H	Store 70H in reg. R2
0014	L2	MOV A,@R0	Copying value from memory address to A
0015		MOV B,A	Copying A to reg. B
0017		MUL AB	Multiplying A & B
0018		MOV @R1,A	Copying A to memory
0019		MOV A,#02H	Store 02H in Accumulator
001B		MOV B,@R0	Copying value from memory address to B
001D		MUL AB	Multiplying A & B

001E		ADD A,#05H	Adding 05H To Accumulator
0020		ADD A,@R1	Adding content from memory to A
0021		MOV @R1,A	Copying A to memory address
0022		INC R0	Increment R0
0023		INC R1	Increment R1
0024		DJNZ R2,L2	If R2 ≠ 0, jump to L2
0026	HERE	JMP HERE	End of program

6. Write a program to find the number of zeros in register R2

Memory Location	Label	Instruction	Comment
0000		MOV R2,#0AH	Store 0AH value in reg. R2
0002		MOV B,#00H	Initialize reg. B with 00H
0005		MOV A,R2	Copy the content from R2 to A
0006		MOV R3,#08H	Set the counter to 08H
0008	L2	RRC A	Rotate A to check no. Of zeroes
0009		JC L1	If carry=1, jump to L1
000B		INC B	If carry =0, increment reg. B
000D	L1	DJNZ R3,L2	If R3 not equal to 0, jump to L2
000F		MOV R1,B	Store the answer in reg. R1
0011	HERE	SJMP HERE	End of program

7. Write a program to check if the accumulator is divisible by 8.

Memory Location	Label	Instruction	Comment
0000		MOV A,#10H	Store a number in Accumulator
0002		MOV B,#08H	Store 08H in reg. B
0005		DIV AB	Divide A by B

0006		MOV 40H,A	Store the answer at memory address 40H
0008		MOV A,B	Copying remainder in reg. B to A
000A		CJNE A,#00H,L1	If remainder =00H, jump to L1
000D		MOV 41H,#01H	If remainder ≠ 00H, Store 01H at memory address 41H
0010	HERE	SJMP HERE	End of the program
0012	L1	MOV 41H,#00H	If remainder =00H, Store 00H at memory address 41H
0015		JMP HERE	

❖❖❖❖

Syllabus

F.Y.B.Sc. (IT), Sem - II,

Microprocessor & Microcontrollers

Unit I : Internet and WWW

What is Internet? Introduction to Internet and its applications, E-mail, telnet, FTP, e-commerce, video conferencing, e-business. Internet service providers, domain name server, internet address

World Wide Web (WWW)

World Wide Web and its evolution, uniform resource locator (URL), browsers – internet explorer, netscape navigator, opera, firefox, chrome, mozilla, search engine, web saver-apache, IIS, proxy server, HTTP protocol

Unit II : HTML and Graphics

HTML Tag Reference, global Attributes, Event Handlers, Document Structure Tags, Formatting Tags, text Level formatting, Block Level formatting, List Tags, Hyperlink tags, Images and Image maps, Table tags, Form Tags, Frame Tags, Executable content tags

Imagemaps

What are Imagemaps? Client-side Imagemaps, Server-side Imagemaps, Using Server-slide and Client-side Imagemaps together, Alternative text for Imagemaps,

Tables

Introduction to HTML tables and theirstructure, The table tags, Alignment, Aligning Entire Table, Alignment within a row, Alignment within a cell, Attributes, Content Summary, Background Color, Adding a Caption, Setting the width, Adding a border, Spacing within a cell, Spacing between the cells, Spanning multiple rows or columns, Elements that can be placed in a table, Table Sections and column properties, Tables as a design tool.

Frames

Introduction to Frames, Applications, Frames document, The <FRAMESET> tag, Nesting <FRAMESET> tag, Placing content in frames with the <FRAME> tag, Targeting named frames, Creating floating frames, Using Hidden frames,

Forms

Creating Forms, The <FORM> tag, Named Input fields, The <INPUT> tag, Multiple lines text windows, Drop down and list boxes, Hidden, Text, Test Area, Password, File Upload, Button, Submit, Reset, radio, Checkbox. Select, Option, Forms and Scripling, Action Buttons, Labelling input files, Grouping related fields, Disabled and read-only fields, Form field event handlers, Passing form data.

Style Sheets

What are style sheets? Why are style sheets valuable? Different approaches to style sheets, Using Multiple approaches, Linking to style information in s separate file, Setting up style information, Using the <LINK> tag, Embedded style information, Using <STYLE> tag, Inline style information.

Unit III : Java Script

Introduction, Client-Side JavaScript, Server-Side Java Script, Java Script Objects, Java Script Security.

Operators

Assignment, Operators, Comparison Operators, Arithmetic Operators, % (Modulus), ++ (Increment), -- (Decreemnt), - (Unary Negation), Logical Operators, Short-Circuit Evaluation, String Operators, Special Operators, : (Conditional operator), (Comma operator), delete, new, this, void

Statements

Break, comment, continue, delete, do…while, export, for, for….in, function, if….else, import, labeled, return, switch, var, while, with,

Core JavaScript (Properties and Methods of Each)

Array, Boolean, Date, Function, Math, Number. Object, String, resExp

Document and its associated objects

Document, Link, Area, Anchor, Image. Applet, Layer

Events and Event Handlers

General Information about Events, Defining Even Handlers, event, onAbort, onBlur, onChange, onClick, onDblClick, ondragDrop, onError, onFocus, onKeyDown, onKeyPress, onKeyUp. onLoad, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onMove, onReset, onResize, onSelect, onSubmit, onUnload

Unit IV : XML

Introduction to XML, Anatomy of an XML document, Creating XML Documents, Creating XML DTDs, XML Schemas, XSL.

Unit V : PHP

Why PHP and MySQL?, Server-side web scripting, Installing PHP, Adding PHP to HTML, Syntax and Variables, Passing information between pages, Strings, Arrays and Array Functions, Numbers, Basic PHP errors/problems.

Unit VI : Advanced PHP and MySQL

PHP/MySQL Functions, Displaying queries in tables, Building Forms from queries, String and Regular Expressions, Sessions, Cookies and HTTP, Type and Type Conversions, E-Mail

Term Work and tutorial

Should contain minimum 5 assignments and two class tests

Practical : Should contain minimum 8 experiments

List of Practicals :

1. Design a web page using different text formatting tags
2. Design a web page with links to different pages and allow navigation between pages.
3. Design a web page with Imagemaps
4. Design a web page with different tables. Design a webpage suing table so that the content appeared well placed.
5. Design a web page using frames.
6. Design a web page with a form that uses all types of controls.
7. Design a website using style sheets so that the pages have uniform style.
8. Using Java Script design a web page that prints factorial / Fibonacci series / any given series.
9. Design a form with a test box and a command button. Using Java script write a program whether the number entered in the text box is a prime number or not.
10. Design a form and validate all the controls placed on the form using Java Script.
11. Design a DTD, corresponding XML document and display it in browser using CSS.
12. Design an XML document and display it in browser using XSL.
13. Design XML Schema and corresponding XML document.
14. Design a php page to process a form.
15. Design a php page for authenticating a user.
16. Design a complete dynamic website with all validations.

❖❖❖❖

Thank You

educlash.com
Just another Way To Learn