Microprocessadores - 2010-1 5/3/2010

Aula 14: Controle de Fluxo do Programa

Professor: Eraldo Silveira e Silva eraldo.silveira@gmail.com

1 Objetivos

- Apresentar o conceito de mudança de fluxo de execução do programa;
- Apresentar as isntruções de mudança de fluxo no 8051;
- Conceituar pilha e subrotinas;
- Incentivar a estrutura de programas na forma de subrotinas.

2 O Controle de Fluxo

O fluxo de execução normal de um programa é basicamente a execução da sequência de instruções, uma após a outra. Ao ler uma instrução da memória de programa a CPU incrementa o contador de programa conforme o tamanho da instrução lida. Desta forma ele armazena ("lembra") o endereço da próxima instrução a ser lida e executada.

O poder de execução das CPUs está, no entanto, na sua capacidade de desviar o fluxo normal, seja incondicionalmente seja segundo uma condição. Desta forma, pode-se executar laços (*loops*) que possibilitam a execução de tarefas repetitivas.

3 As instruções de jump e endereços absolutos e relativos

As instruções de *jump* (salto) permitem mudar o valor do contador de programa de forma que a próxima instrução a ser executada não seja mais a próxima instrução e sim um valor fornecido na instrução.

Existem três possibilidades para o fornecimento deste endereço. A primeira está associada a instruções que podem fornecer o endereço absoluto (16bits) que substituirá integralmente o contador de programa. Estes dois bytes de endereço acompanham a instrução de jump e permitem endereçar de 0000H a FFFFH.

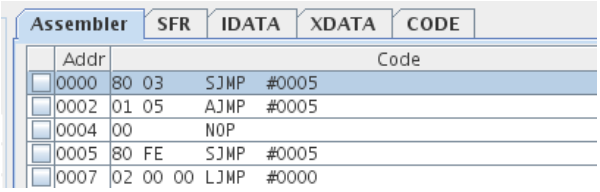
Na segunda possibilidade, é fornecido um endereço relativo (*offset*) de 8 bits. Este valor será será somado ao contador de programa de forma que o endereço da próxima instrução esteja 128 bytes para trás do PC ou 127 bytes para frente.

A terceira possibilidade é o uso de endereçamento absoluto de *short range* baseado no conceito de páginas. Pode-se imaginar a divisão da memória de código de 64K em 32 páginas de 2K. Neste caso, os 5 bits superiores de um endereço de 16 bits correspondem a uma página. Quando uma instrução de jump usa o conceito de endereço absoluto de short range, ela fornece os 11 bits inferiores a serem colocados no contador de programa e os 5 bits posteriores correspodem a página da próxima instrução (os 5 bits superiores do contador de programa). Note que este tipo de endereçamento possui limitações quando a instrução de jump está no limite de uma página.

4 As instruções de jump disponíveis

As instruções disponíveis podem ser organizadas da seguinte forma:

- Jumps Condicionais por teste de bit:



Assembler | SFR | IDATA | XDATA | CODE

Addr	Code		
0000	80 03	SJMP	#0005
0002	01 05	AJMP	#0005
0004	00	NOP	
0005	80 FE	SJMP	#0005
0007	02 00 00	LJMP	#0000

Figura 1: Exemplo de Código Gerado para diferentes Jumps

```
JC label
JNC label
JB bit,label
JNB bit,label
JBC bit,label
```

Estas instruções podem ser usadas em testes de CARRY para operações aritméticas, além das aplicações que já foram vistas anteriormente.

- Jumps Condicionais por teste de byte:

```
CJNE A,direto,label
CJNE A,#imediato,label
CJNE Rn,#imediato,label
CJNE @Rp,#imediato,label
DJNZ Rn,label
DJNZ direto,label
JZ label
JNZ label
```

O label é um endereço relativo de 8 bits. Nas instruções $CJNE$ o flag CARRY é setado caso o conteúdo do primeiro operando seja menor que o segundo operando. Nas demais instruções o CARRY não é afetado.

- Jumps Incondicionais:

```
JMP @A+DPTR
AJMP addr11
LJMP addr16
SJMP addr8
```

Simule a execução do programa abaixo e confira o uso das instruções de *jump* e de endereços relativos e absolutos de *long* e *short range*.

```
 1:          N      0000            org 0H
 2:    0000  21 00                  ajmp BGN
 3:          N      0100            org 100h
 4:    0100  74 30          BGN:    mov A,#30H
 5:    0102  75 50 00               mov 50H,#00H
 6:    0105  B5 50 02       AGN:    cjne A,50H,AEQ
 7:    0108  80 04                  sjmp NXT
 8:    010A  D5 50 F8       AEQ:    djnz 50H,AGN
 9:    010D  00                     nop
10:    010E  78 FF          NXT:    mov R0,#0FFH
11:    0110  D8 FE          DWN:    djnz R0,DWN
12:    0112  E8                     mov A,R0
13:    0113  70 03                  jnz ABIG
14:    0115  02 10 00               ljmp  AZR0
15:    0118  00             ABIG:   nop
16:          N      1000            org 1000H
17:    1000  74 08          AZR0:   mov A,#08H
```

```
18:     1002  90 10 00                mov DPTR,#1000H
19:     1005  73                      jmp @A+DPTR
20:     1006  00                      nop
21:     1007  00                      nop
22:     1008  01 00           HERE:   ajmp AZR0
23:                                   END
```

5 A pilha

Embora a pilha não esteja diretamente ligada ao fluxo de execução do programa, aproveitaremos este momento para explicá-la pelo fato de que ela é necessária ao funcionamento das subrotinas.

A pilha é uma área da memória RAM interna que será operada por algumas instruções que usarão como apoio o registrador SP.

Seja a figura abaixo:

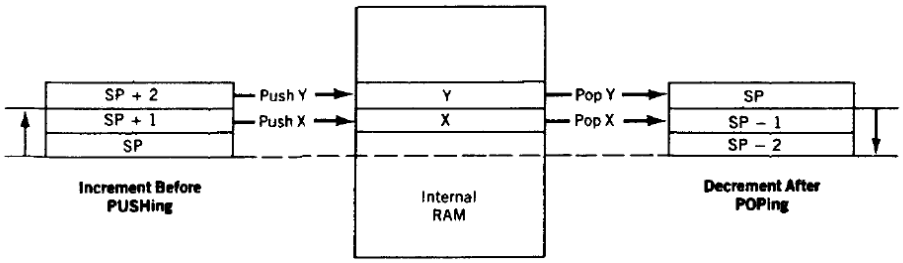


Figura 2: Estrutura da pilha

As instruções de acesso a pilha são o $PUSHaddr8$ e o $POPaddr8$ onde $addr8$ é um endereço de 8 bits da RAM interna. Note que os assembladores permitirão executar instruções como $PUSHB$ ou $PUSHA$.

A operação $PUSH$ é basicamente incrementar o SP em uma unidade e mover o conteúdo de $addr8$ para a memória apontada por SP. A operação POP retira o conteúdo de SP colocando-o na posição $addr8$.

6 As subrotinas

Uma forma interessante de construir programas é organizá-los em pedaços de código que podem ser chamados a qualquer momento e, no fim deste, retornar ao próximo ponto de onde foi invocado.

As instruções básicas para chamada de subrotina são o $LCALL$ e o $ACALL$. O retorno é realizado com RET. Verifique como estas instruções operam com a pilha através da Fig.3.

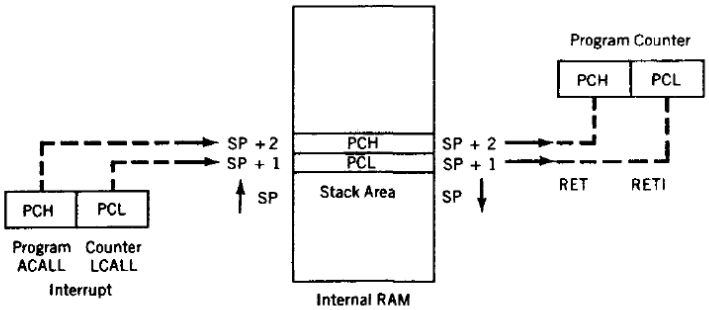


Figura 3: Estrutura da pilha

7 Exercícios

1. Faça uma subrotina para zerar n posições de memória RAM interna onde n é passado como parâmetro em R0 e o início da área em R1.

2. Faça uma subrotina para copiar n posições de memória RAM interna para outra área da RAM interna. Passe os parâmetros em R0, R1 e R2.