ریزپردازنده

دانشکده کامپیوتر دانشگاه یزد

نیم‌سال دوم تحصیلی ۹۷-۹۶

ارائه‌دهنده : پریسا استواری

مجموعه دستورات 8051

انواع دستورات

- دستورات محاسباتی (Arithmetic)
- دستورات منطقی (Logic)
- دستورات انتقال اطلاعات (Data Transfer)
- دستورات بر روی بیت یا متغیرهای بولین (Boolean Variables)
- دستورات کنترل یا انشعاب برنامه (Program Branching)

دستورات پرش بدون شرط

دستور SJMP

SJMP rel

- Short Jump
- این دستور از آدرس‌دهی نسبی استفاده می‌کند. (یادآوری اسلاید ۳)

- این دستور ۲ بایتی است.
  - بایت اول کد اجرا (opcode)
  - بایت دوم آفست پرش که یک عدد علامت‌دار ۸ بیتی است. آدرس مقصد پرش در ۱۲۷ بایت پایین تر یا ۱۲۸ بایت بالاتر نسبت به مقدار فعلی PC خواهد بود.

- عملکرد :
  - PC <- PC + 2
  - بایت دوم دستور + PC <- PC

دستور SJMP

آفست + آدرس منبع = آدرس مقصد

آدرس منبع – آدرس مقصد = آفست

- مثال : برچسب HERE در آدرس 0123H قرار دارد. و دستور زیر در آدرس 0100H. بایت دوم دستور SJMP که حاوی آفست پرش است چه مقداری خواهد داشت؟

SJMP HERE

- حل :

0123H – 0102H = 21H

دستور SJMP

آفست + آدرس منبع = آدرس مقصد

آدرس منبع – آدرس مقصد = آفست

- مثال : اگر دستور SJMP HERE در آدرس 0300H قرار داشته باشد و بایت دوم دستور SJMP برابر با آفست A6H باشد، آدرس مقصد کجا است؟

- حل :

A6H = -5AH

0302H – 5AH = 02A8

دستور AJMP

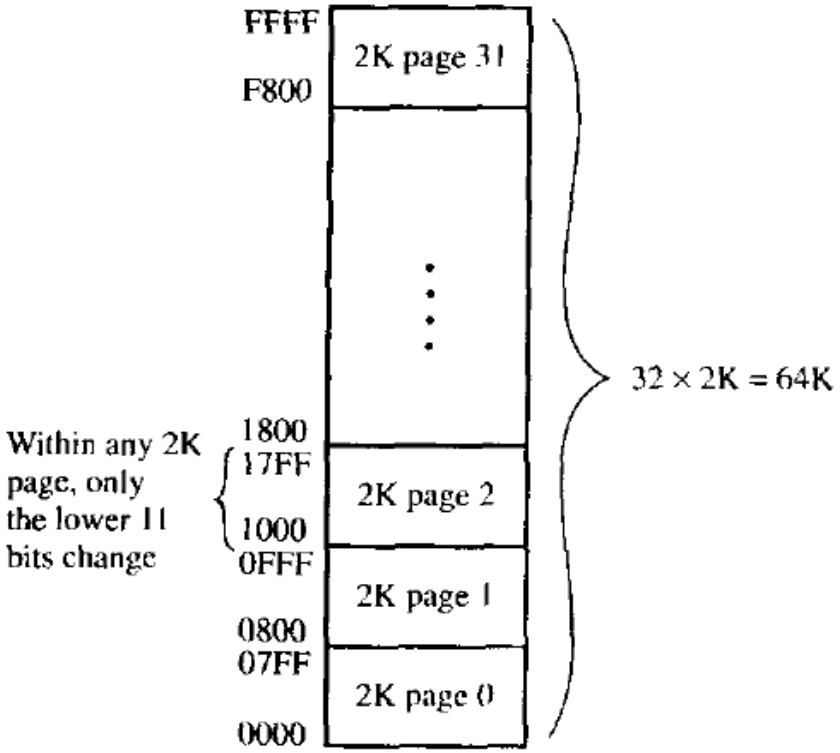
AJMP adr11

- Absolute Jump
- این دستور از آدرس‌دهی مطلق استفاده می‌کند. (یادآوری اسلاید ۳)
  - مقصد پرش تنها می‌تواند در صفحه‌ی 2K بایتی باشد.

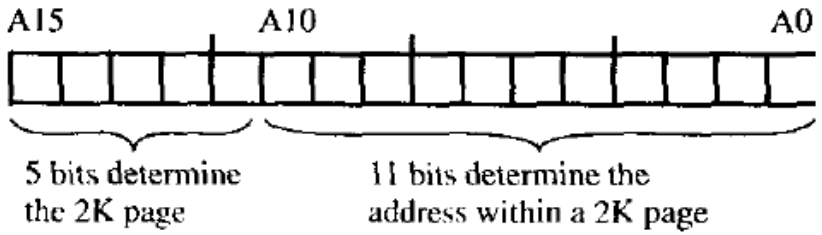
- این دستور دو بایتی است. ۱۱ بیت کم ارزش‌تر آدرس مقصد در کد دستور قرار می‌گیرد.
  - هنگام اجرا، ۵ بیت با ارزش‌تر PC تغییر نمی‌کند، ۱۱بیت آدرس که در کد دستور است در ۱۱ بیت کم ارزش‌تر PC کپی می‌شود.

دستور AJMP

- در داخل صفحه با طول 2K فقط ۱۱ بیت کم ارزش‌تر آدرس تغییر می‌کند.
- ۵ بیت با ارزش‌تر PC نشان‌دهنده‌ی شماره صفحه و ۱۱ بیت کم ارزش‌تر آن نشان-دهنده آدرس داخل آن صفحه است.
- عملکرد :
- PC <- PC + 2
- آدرس داخل صفحه -> PC10-PC0



(a) 64K memory map divided into 32 2K pages



(b) The upper 5 bits in the program counter remain the same. The lower bits are replaced by the bits supplied in the instruction.

دستور AJMP

- مثال : دستور AJMP HERE در آدرس 0123H قرار دارد و برچسب HERE در آدرس 0345H. آدرس ۱۱ بیتی که در کد دستور AJMP قرار می‌گیرد چند است؟

0125H = 0000 0001 0010 0101
0345H = 0000 0011 0100 0101

- کد دستور AJMP HERE (61H 45H):

01100001
01000101

دستور LJMP

LJMP adr16

- Long Jump
- این دستور از آدرس‌دهی بلند استفاده می‌کند. (یادآوری اسلاید ۳)
- طول این دستور ۳ بایت است و با استفاده از آن می‌توان به کل فضای حافظه ROM پرش کرد. از 0000H تا FFFFH
  - یک بایت برای کد دستور و ۲ بایت برای آدرس ۱۶ بیتی

- آدرس ۱۶ بیتی مقصد که در بایت دوم و سوم دستور وجود دارد به طور کامل در PC کپی می‌شود.

- مثال : اگر دستور LJMP HERE در آدرس 1234H باشد و برچسب HERE در آدرس 0123H، کد اجرای این دستور به صورت زیر خواهد بود :

00000010 00000001 00100011

دستور JMP

JMP label

- اگر به طور کلی برای هر پرش دستور JMP را بنویسیم، مترجم اسمبلر، خود متناسب با آدرس مقصد، یکی از دستورات SJMP، AJMP و یا LJMP را به جای دستور JMP قرار می‌دهد.

دستور JMP

JMP @A+DPTR

- Jump Table

- این دستور برای جدول پرش به کار می‌رود. یعنی بسته به حالت‌های مختلف، پرش به نقاط متفاوت برنامه صورت می‌گیرد.
- معمولا ثبات DPTR با آدرس شروع جدول پرش بار می‌شود و مقدار A به عنوان اندیس به کار می‌رود.
- این دستور عدد ۸ بیتی محتوای A را به عدد ۱۶ بیتی محتوای DPTR اضافه کرده و حاصل را در PC قرار می‌دهد تا آدرس دستور بعدی که باید واکشی شود مشخص گردد.
- طول این دستور ۱ بایت است.

دستور JMP

- مثال : اگر ۵ حالت پرش به نقاط مختلف برنامه لازم باشد، مقدار A با مقادیر 0 تا 4 بار می‌شود تا پرش مطابق دستورات زیر به نقاط مختلف انجام شود.

```
                MOV DPTR, #JUMP_TABLE
                MOV A,#INDEX_NUMBER
                RL A
                JMP @A+DPTR
                ……
JUMP_TABLE :    AJMP CASE0
                AJMP CASE1
                AJMP CASE2
                AJMP CASE3
                ……
CASE0 :         …..
CASE1 :         …..
CASE2 :         …..
CASE3 :         …..
```

آدرس جدول پرش را در DPTR قرار می‌دهد.

شماره اندیس را به A منتقل می‌کند.

عدد اندیس را از 0 تا 4 به 0 تا 8 تبدیل می‌کند.
این عمل برای این است که طول دستورات AJMP هر کدام ۲ بایتی است.

دستور JMP

- در مثال قبل، اگر جدول پرش از خانه 8100H حافظه‌ی ROM شروع شود و دارای مقادیر زیر باشد،
- الف: ابتدا و انتهای آدرس صفحه 2K بایتی حافظه که دستورات در آن قرار دارد چیست؟
- ب: CASE0 و CASE3 هر کدام از چه آدرسی شروع می‌شوند؟

آدرس	محتوا
8100	01
8101	B8
8102	01
8103	43
8104	41
8105	76
8106	E1
8107	F0

- حل :
- الف: ابتدا و انتهای صفحه 2K بایتی برابر است با 8000H و 87FFH
- ب: برای CASE3 داریم کد اجرای آن E1 F0 است.
- E1 F0 = 1110 0001 1111 0000
- 1000 0111 1111 0000 = 87F0H = آدرس سابروتین CASE3

دستور JMP

- مثال : در دستورات زیر، مقدار A باید چه مقادیری باشد تا پرش به LABEL0 تا LABEL3 انجام شود.
- اگر بخواهیم با اندیس 0 تا 3 برای A پرش‌ها صورت بگیرد، چه دستوراتی باید به برنامه اضافه گردد؟

```
                MOV DPTR, #JMP_TBL
                MOV A, #INDEX_NUMBER
                ?
                JMP @A+DPTR
                ……..
JMP_TBL:        LJMP LABEL0
                LJMP LABEL1
                LJMP LABEL2
                LJMP LABEL3
```

برای پرش به LABEL0 تا LABEL3 باید مقدار A برابر با 0، 3، 6 و 9 باشد.
زیرا دستور LJMP سه بایتی است.

دستور JMP

- مثال : در دستورات زیر، مقدار A باید چه مقادیری باشد تا پرش به LABEL0 تا LABEL3 انجام شود.
- اگر بخواهیم با اندیس 0 تا 3 برای A پرش‌ها صورت بگیرد، چه دستوراتی باید به برنامه اضافه گردد؟

```
                MOV DPTR, #JMP_TBL
                MOV A, #INDEX_NUMBER
                MOV B, #3
                MUL AB
                JMP @A+DPTR
                ……..
JMP_TBL:        LJMP LABEL0
                LJMP LABEL1
                LJMP LABEL2
                LJMP LABEL3
```

دستورات پرش

دستور	ترجمه دستور	تعداد بایت	تعداد سیکل ماشین
SJMP rel	10000000 eeeeeeee	2	2
AJMP adr11	aaa00001 aaaaaaaa	2	2
LJMP adr16	00000010 aaaaaaaa aaaaaaaa	3	2
JMP @A+DPTR	01110011	1	2

دستورات پرش شرطی

- میکرو کنترلر 8051 دارای تعدادی دستورات پرش شرطی است که تمام آنها به آدرس نسبی در محدوده 127+ و 128- بایت بعد از دستور مذکور پرش می‌کنند.
- آدرس مقصد معمولا توسط برچسب (label) مشخص می‌شود.

دستور JZ

JZ rel

- Jump Zero

- اگر محتوای A برابر با 0 باشد، به آدرس مقصد پرش می‌کند، در غیر اینصورت دستور بعدی را اجرا می‌کند.

- این دستور دو بایت است. بایت دوم دستور آفست نسبی پرش است.
  - (PC) <- (PC) + 2
  - اگر مقدار A برابر با 0 بود :
  - بایت دوم دستور + (PC) <- (PC)

- مثال :

```
        MOV A, R1
        JZ SKIP
        ...
SKIP :  ...
```

دستور JNZ

JNZ rel

- Jump Not Zero

- اگر محتوای A مخالف 0 باشد، به آدرس مقصد پرش می‌کند، در غیر اینصورت دستور بعدی را اجرا می‌کند.

- این دستور دو بایت است. بایت دوم دستور آفست نسبی پرش است.
  - (PC) <- (PC) + 2
  - اگر مقدار A برابر با 0 نبود :
  - بایت دوم دستور + (PC) <- (PC)

- مثال : اگر مقدار R5 صفر است، مقدار 55H را در آن بریز.

```
        MOV A, R5
        JNZ NEXT
        MOV R5, #55H
        ...
NEXT:   ...
```

دستور CJNE

CJNE dest, src, adr

- Compare and Jump if Not Equal

- این دستور مقدار دو اپرند dest و src را با هم مقایسه می‌کند و اگر برابر نبودند به آدرس adr پرش می‌کند.

- اگر محتوای اپرند مقصد کوچکتر از محتوای اپرند منبع باشد، بیت پرچم نقلی C برابر با 1 می‌شود.

- اگر اپرند مقصد A باشد، اپرند منبع می‌تواند آدرس‌دهی مستقیم یا آدرس‌دهی بلافاصله (عدد ثابت) باشد.
- اگر اپرند مقصد یکی از ثبات‌های R0 تا R7 باشد یا یک خانه‌ی حافظه با آدرس‌دهی غیر مستقیم باشد، اپرند منبع باید عدد ثابت باشد.

دستور CJNE

- مثال : این دستورات تا زمانی که مقدار پورت P1 برابر با مقدار A نشود در همین دستور صبر می‌کند و زمانی که مساوی شد دستورات بعد اجرا می‌شود.

```
WAIT:   CJNE A, P1, WAIT
```

- یا می‌توان خط بالا را بدین صورت نیز نوشت :

```
CJNE A, P1, $
```

- علامت دلار در زبان اسمبلی میکروکنترلر به معنای آدرس فعلی دستور است.

دستور CJNE

- مثال : در این دستور محتوای R7 با عدد 60H مقایسه می‌شود و مشخص می‌گردد R7 کوچکتر، بزرگتر یا مساوی عدد 60H است.

```
                CJNE R7, #60H, NOT_EQ
EQ:             ...
                SJMP EXIT
NOT_EQ:         JC REG_LOW
REG_HIGH:       ...
                SJMP EXIT
REG_LOW:        ...
EXIT:           ...
```

دستور DJNZ

DJNZ operand, adr

- Decrement and Jump Not Zero

- این دستور برای عملیات حلقه استفاده می‌شود.
- این دستور از اپراند یک واحد کم می‌کند، اگر نتیجه 0 نباشد، به آدرس مقصد پرش می‌کند.
  - اپرند می‌تواند یکی از ثبات‌های R0 تا R7 یا آدرس مستقیم خانه حافظه باشد.

اگر مقدار 0 در R2 ریخته شود، حلقه ۲۵۶ بار تکرار می‌شود.
اگر مقدار FFH در R2 ریخته شود، حلقه ۲۵۵ بار تکرار می‌شود.

```
;This program adds value 3 to the ACC ten times
        MOV   A,#0      ;A=0, clear ACC
        MOV   R2,#10    ;load counter R2=10
AGAIN:  ADD   A,#03     ;add 03 to ACC
        DJNZ  R2,AGAIN  ;repeat until R2=0,10 times
        MOV   R5,A      ;save A in R5
```

حلقه های تو در تو

- اگر به حلقه ای نیاز داشتیم که بیشتر از ۲۵۶ بار تکرار شود، باید از حلقه‌های تو در تو (nested loop) استفاده کنیم.

- مثال : برنامه‌ای بنویسید که مقدار #55H را به A ریخته و محتوای A را ۷۰۰ بار مکمل کند.

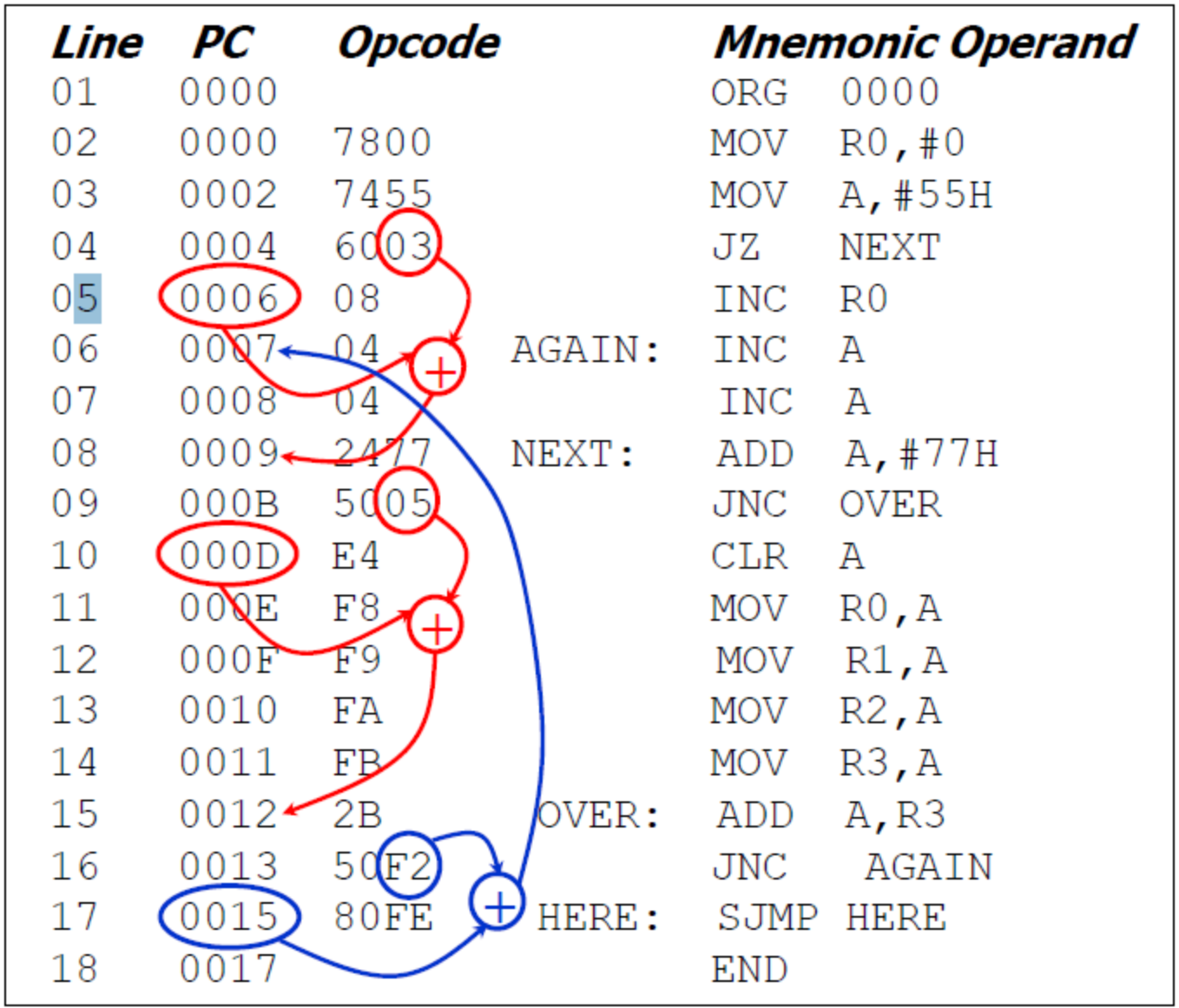
```
        MOV  A,#55H   ;A=55H
        MOV  R3,#10   ;R3=10, outer loop count
NEXT:   MOV  R2,#70   ;R2=70, inner loop count
AGAIN:  CPL  A        ;complement A register
        DJNZ R2,AGAIN ;repeat it 70 times
        DJNZ R3,NEXT
```

دستورات پرش شرطی

دستور	ترجمه دستور	تعداد بایت	تعداد سیکل ماشین
JZ rel	01010000 eeeeeeee	2	2
JNZ rel	01110000 eeeeeeee	2	2
DJNZ Rn, rel	11011rrr eeeeeeee	2	2
DJNZ direct, rel	11010101 aaaaaaaa eeeeeeee	3	2
CJNE A, direct, rel	10110101 aaaaaaaa eeeeeeee	3	2
CJNE A, #data, rel	10110100 dddddddd eeeeeeee	3	2
CJNE Rn, #data, rel	10111rrr dddddddd eeeeeeee	3	2
CJNE @Ri, #data, rel	1011011i dddddddd eeeeeeee	3	2

- یادآوردی : دستورات زیر برای چک کردن بیت و پرش به کار می‌رفتند.
  - JC JNC JB JNB JBC
- تمام دستورات پرش شرطی، پرش کوتاه هستند.
  - آدرس پرش باید در محدوده 127+ و 128- از مقدار فعلی PC باشد.
  - بایت آخر این دستورات با محتوای PC جمع می‌شود.

محاسبه آدرس پرش کوتاه



Line	PC	Opcode		Mnemonic	Operand
01	0000			ORG	0000
02	0000	7800		MOV	R0,#0
03	0002	7455		MOV	A,#55H
04	0004	6003		JZ	NEXT
05	0006	08		INC	R0
06	0007	04	AGAIN:	INC	A
07	0008	04		INC	A
08	0009	2477	NEXT:	ADD	A,#77H
09	000B	5005		JNC	OVER
10	000D	E4		CLR	A
11	000E	F8		MOV	R0,A
12	000F	F9		MOV	R1,A
13	0010	FA		MOV	R2,A
14	0011	FB		MOV	R3,A
15	0012	2B	OVER:	ADD	A,R3
16	0013	50F2		JNC	AGAIN
17	0015	80FE	HERE:	SJMP	HERE
18	0017			END	

دستورات فراخوانی سابروتین

- از سابروتین برای انجام یک کار به صورت تکراری استفاده می‌شود.
  - در اینصورت در حافظه‌ی مصرف‌شده صرفه‌جویی می‌شود.

- اگر دستور CALL در برنامه به کار برده شود، محتوای PC که آدرس دستور بعدی CALL است در حافظه پشته قرار می‌گیرد.
  - ابتدا بایت کم ارزش‌تر آدرس و سپس بایت پر ارزش‌تر آدرس در حافظه پشته قرار می‌گیرد.
- سپس مقدار PC با آدرس سابروتین که در دستور قرار دارد بار می‌شود.
- در انتهای هر سابروتین باید دستور برگشت از سابروتین RET قرار بگیرد تا دستور بعد از CALL در برنامه اجرا شود.
- دستور RET دو بایت آدرس برگشت که قبلا توسط دستور CALL در پشته ذخیره شده بود را به PC بار می‌کند.

دستورات CALL

- دو نوع دستور CALL داریم.

- ACALL
  - Absolute Call
  - از آدرس‌دهی مطلق استفاده می‌کند.
  - دستور ۲ بایتی است. که ۱۱ بیت آن برای آدرس سابروتین درون صفحه 2K بایتی استفاده می‌شود.

- LCALL
  - Long Call
  - از آدرس‌دهی بلند استفاده می‌کند.
  - دستور ۳ بایتی است.
  - بایت اول opcode است و بایت دوم و سوم برای آدرس ۱۶ بیتی سابروتین استفاده می‌شود.

دستورات RET

RET

- Return
- دستور RET باعث می‌شود بایت پر ارزش‌تر و کم ارزش‌تر PC از حافظه پشته بازیابی شود و در نتیجه دستور بلافاصله بعد از ACALL یا LCALL اجرا گردد.

RETI

- Return Interrupt
- دستور RETI بایت پر ارزش‌تر و کم ارزش‌تر آدرس بازگشت به برنامه اصلی را از پشته بازیابی می‌کند و در PC قرار می‌دهد تا برنامه اصلی مجددا اجرا گردد.
- همچنین مدار وقفه را در حالتی قرار می‌دهد که آماده پذیرش وقفه جدید باشد.

مثال سابروتین

```
        ORG 0
BACK:   MOV A, #55H
        MOV P1, A
        LCALL DELAY
        MOV A, #0AAH
        MOV P1, A
        LCALL DELAY
        SJMP BACK

        ORG 300H
DELAY:  MOV R5, #0FFH
        DJNZ R5, $
        RET
```

برای اجرای دستور CALL، آدرس دستور بعد یعنی دستور MOV A, #0AAH در استک قرار می‌گیرد و میکروکنترلر از دستور موجود در خانه 300H حافظه اجرا می‌شود.

این حلقه ۲۵۵ بار اجرا می‌شود و یک تاخیر نرم افزاری ایجاد می‌کند.

هنگامی که مقدار R5 صفر می‌شود، دستور RET اجرا می‌شود که آدرس دستوری که در استک است را POP کرده و آن را در PC قرار می‌دهد و کار از دستور بعد از CALL ادامه می یابد.

مثال سابروتین

```
001 0000                ORG   0
002 0000 7455    BACK:  MOV   A,#55H   ;load A with 55H
003 0002 F590           MOV   P1,A     ;send 55H to p1
004 0004 120300         LCALL DELAY    ;time delay
005 0007 74AA           MOV   A,#0AAH  ;load A with AAH
006 0009 F590           MOV   P1,A     ;send AAH to p1
007 000B 120300         LCALL DELAY
008 000E 80F0           SJMP  BACK     ;keep doing this
009 0010
010 0010 ;-------this is the delay subroutine------
011 0300                ORG   300H
012 0300         DELAY:
013 0300 7DFF           MOV   R5,#0FFH ;R5=255
014 0302 DDFE    AGAIN: DJNZ  R5,AGAIN ;stay here
015 0304 22             RET            ;return to caller
016 0305                END             ;end of asm file
```

محتوای استک بعد از فراخوانی اولین LCALL

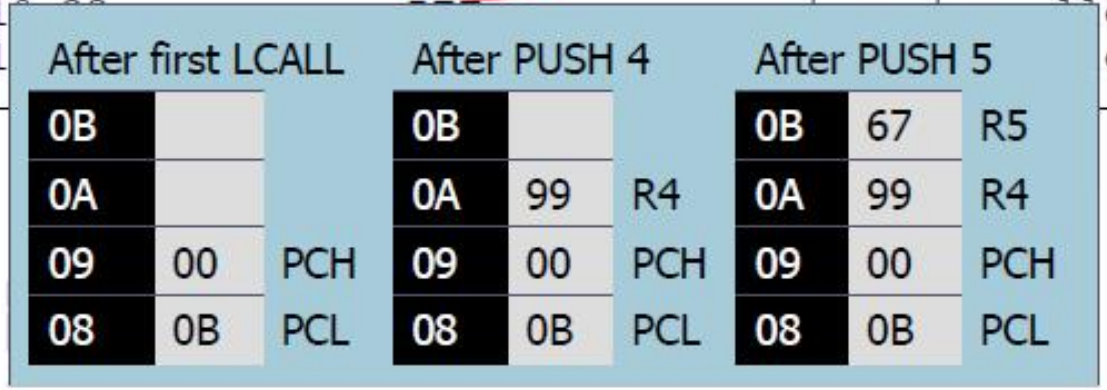
0A	
09	00
08	07

SP = 09

ابتدا بایت کم ارزش‌تر آدرس بازگشت در استک ذخیره می‌شود، سپس بایت پر ارزش‌تر

```
01 0000                  ORG  0
02 0000 7455   BACK:     MOV  A,#55H  ;load A with 55H
03 0002 F590             MOV  P1,A    ;send 55H to p1
04 0004 7C99             MOV  R4,#99H
05 0006 7D67             MOV  R5,#67H
06 0008 120300           LCALL DELAY  ;time delay
07 000B 74AA             MOV  A,#0AAH ;load A with AA
08 000D F590             MOV  P1,A    ;send AAH to p1
09 000F 120300           LCALL DELAY
10 0012 80EC             SJMP  BACK   ;keeping doing
   this
11 0014 ;-------this is the delay subroutine------
12 0300                  ORG  300H
13 0300 C004   DELAY:    PUSH 4       ;push R4
14 0302 C005             PUSH 5       ;push R5
   0304 7CFF             MOV  R4,#0FFH;R4=FFH
   0306 7DFF   NEXT:     MOV  R5,#0FFH;R5=FFH
   0308 DDFE   AGAIN:    DJNZ R5,AGAIN
   030A DCFA             DJNZ R4,NEXT
   030C D005             POP  5       ;POP into R5
   030E D004             POP  4       ;POP into R4
   031                                          er
22 031                                          e
```

باید دقت شود در بدنه سابروتین، باید تعداد PUSH و POP ها برابر باشد.



```
;MAIN program calling subroutines
        ORG   0
MAIN:   LCALL           SUBR_1
        LCALL           SUBR_2
        LCALL           SUBR_3

HERE:   SJMP            HERE
;-----------end of MAIN

SUBR_1: ...
        ...
        RET
;-----------end of subroutine1

SUBR_2: ...
        ...
        RET
;-----------end of subroutine2

SUBR_3: ...
        ...
        RET
;-----------end of subroutine3
        END             ;end of the asm file
```

مرسوم است که در بدنه‌ی اصلی برنامه، چندین سابروتین فراخوانی شود.

این کار اجازه می‌دهد هر سابروتین به صورت ماجول جداگانه‌ای نوشته و تست شود و سپس در برنامه‌ی اصلی، ماجول‌های تولید شده کنار یکدیگر قرار گیرند.

این روش در نوشتن برنامه‌های بزرگ بسیار موثر است.

دستور CALL

- تفاوت دستور ACALL و LCALL
  - آدرس مقصد در دستور ACALL باید در صفحه‌ی 2K بایتی همان دستور باشد.
  - آدرس مقصد در دستور LCALL می‌تواند هر کجای حافظه 64K بایتی ROM باشد.
- استفاده از ACALL به جای LCALL تنها در چندین بایت صرفه‌جویی خواهد شد.
- می‌توان در برنامه نویسی از دستور اسمبلر CALL استفاده کرد.
  - اسمبلر بسته به آدرس مقصد، دستور ACALL یا LCALL را جایگزین دستور CALL می‌نماید.

```
        ORG    0
BACK:   MOV    A,#55H     ;load A with 55H
        MOV    P1,A       ;send 55H to port 1
        LCALL  DELAY      ;time delay
        MOV    A,#0AAH    ;load A with AA (in hex)
        MOV    P1,A       ;send AAH to port 1
        LCALL  DELAY
        SJMP   BACK       ;keep doing this indefinitely
         ...
        END               ;end of asm file
```

برنامه‌ی مشابه برنامه‌ی بالا که از نظر حافظه‌ی کد بهینه‌تر است.

A rewritten program which is more efficiently

```
        ORG    0
        MOV    A,#55H     ;load A with 55H
BACK:   MOV    P1,A       ;send 55H to port 1
        ACALL  DELAY      ;time delay
        CPL    A          ;complement reg A
        SJMP   BACK       ;keep doing this indefinitely
         ...
        END               ;end of asm file
```

دستور NOP

NOP

- No Operation
- این دستور عملیاتی انجام نمی‌دهد.
  - بعد از واکشی این دستور عملیاتی انجام نمی‌شود تنها یک سیکل ماشین وقفه ایجاد می‌شود. سپس دستور بعدی واکشی و اجرا می‌شود.
- کد اجرای این دستور 00H است.

- مثال : می‌خواهیم در بیت ۷ پورت p2 یک پالس پایین به اندازه‌ی ۵ سیکل ماشین ایجاد کنیم.


```
CLR P2.7
NOP
NOP
NOP
NOP
SETB P2.7
```

زمان صفر و یک کردن یک بیت یک سیکل ماشین است، لذا به ۴ NOP نیاز داریم.

دستورات سابروتین و وقفه

دستور	ترجمه دستور	تعداد بایت	تعداد سیکل ماشین
ACALL	aaa10001 aaaaaaaa	2	2
LCALL	00010010 aaaaaaaa aaaaaaaa	3	2
RET	00100010	1	2
RETI	00110010	1	2
NOP	00000000	1	1

تاخیر زمانی

- اجرای هر یک از دستورات در میکروکنترلر تعداد مشخصی سیکل ماشین به طول می‌انجامد.
- طول یک سیکل ماشین وابسته است به فرکانس اسیلاتور متصل به میکروکنترلر.
- در میکروکنترلر 8051 هر سیکل ماشین ۱۲ کلاک اسیلاتور به طول می‌انجامد.
- اگر اسیلاتور متصل به میکروکنترلر 8051 دارای فرکانس 12MHz باشد، هر سیکل ماشین در 8051 1µs به طول می‌کشد.
- مثال : اگر فرکانس اسیلاتور 11.0592MHz باشد، یک سیکل ماشین چند ثانیه است؟

11.0592 / 12 = 921.6 KHz

Machine Cycle is 1 / 921.6 K = 1.085 µs

تاخیر زمانی

- برای یک میکروکنترلر 8051 با اسیلاتور با فرکانس 11.0592 MHz، هر کدام از دستورات زیر در چه زمانی اجرا می‌گردند؟

```
(a) MOV R3,#55 (b) DEC R3 (c) DJNZ R2 target
(d) LJMP (e) SJMP (f) NOP (g) MUL AB
```

Solution:

	Machine cycles	***Time to execute***
(a)	1	$1x1.085\mu s = 1.085\mu s$
(b)	1	$1x1.085\mu s = 1.085\mu s$
(c)	2	$2x1.085\mu s = 2.17\mu s$
(d)	2	$2x1.085\mu s = 2.17\mu s$
(e)	2	$2x1.085\mu s = 2.17\mu s$
(f)	1	$1x1.085\mu s = 1.085\mu s$
(g)	4	$4x1.085\mu s = 4.34\mu s$

محاسبه تاخیر زمانی

Find the size of the delay in following program, if the crystal frequency is 11.0592MHz.

```
        MOV   A,#55H
AGAIN:  MOV   P1,A
        ACALL DELAY
        CPL   A
        SJMP  AGAIN
;---time delay------
DELAY:  MOV   R3,#200
HERE:   DJNZ  R3,HERE
        RET
```

Solution:

```
                        Machine cycle
DELAY:  MOV   R3,#200          1
HERE:   DJNZ  R3,HERE          2
        RET                    2
```

Therefore, $[(200x2)+1+2]x1.085\mu s = 436.255\mu s.$

محاسبه تاخیر زمانی

Find the size of the delay in following program, if the crystal frequency is 11.0592MHz.

```
                          Machine Cycle
DELAY: MOV  R3,#250          1
HERE:  NOP                   1
       NOP                   1
       NOP                   1
       NOP                   1
       DJNZ R3,HERE          2
       RET                   2
```

Solution:

The time delay inside HERE loop is

$[250(1+1+1+1+2)]\times 1.085\,\mu s = 1627.5\,\mu s.$

Adding the two instructions outside loop we have $1627.5\,\mu s + 3 \times 1.085\,\mu s = 1630.755\,\mu s$

محاسبه تاخیر زمانی

		Machine Cycle
DELAY:	MOV R6, #0FFH	1
BACK:	MOV R7, #100	1
HERE:	DJNZ R7, HERE	2
	DJNZ R6, BACK	2
	RET	2

- Solution:
- = 1 + (1 + 100 * 2 + 2) * 255 + 2 = 51768 سیکل ماشین

- اگر هر سیکل ماشین را برابر 1 میکرو ثانیه فرض کنیم، تاخیری که این زیربرنامه ایجاد می‌کند برابر است با 51768 میکرو ثانیه یا حدودا 50 میلی ثانیه.

محاسبه تاخیر زمانی

		Machine Cycle
	MOV R6, #200	1
BACK1:	MOV R7, #100	1
BACK2:	NOP	1
	NOP	1
	DJNZ R7, BACK2	2
	NOP	1
	NOP	1
	DJNZ R6, BACK1	2
	RET	2

- Solution:
- = 1 + (1 + (1 + 1 + 2) * 100 + 1 + 1 + 2) * 200 + 2 = 81003 سیکل ماشین

محاسبه تاخیر زمانی

		Machine Cycle
	MOV R6, #200	1
BACK1:	MOV R7, #100	1
BACK2:	NOP	1
	NOP	1
	DJNZ R7, BACK2	2
	NOP	1
	NOP	1
	DJNZ R6, BACK1	2
	RET	2

- Solution:
- = 1 + (1 + (1 + 1 + 2) * 100 + 1 + 1 + 2) * 200 + 2 = 81003 سیکل ماشین

محاسبه تاخیر زمانی

		Machine Cycle
	MOV R6, #200	1
BACK1:	MOV R7, #100	1
BACK2:	NOP	1
	NOP	1
	DJNZ R7, BACK2	2
	NOP	1
	NOP	1
	DJNZ R6, BACK1	2
	RET	2

- Solution:
- = 1 + (1 + (1 + 1 + 2) * 100 + 1 + 1 + 2) * 200 + 2 = 81003 سیکل ماشین

محاسبه تاخیر زمانی

		Machine Cycle
	MOV R6, #200	1
BACK1:	MOV R7, #100	1
BACK2:	NOP	1
	NOP	1
	DJNZ R7, BACK2	2
	NOP	1
	NOP	1
	DJNZ R6, BACK1	2
	RET	2

- Solution:
- = 1 + (1 + (1 + 1 + 2) * 100 + 1 + 1 + 2) * 200 + 2 = 81003 سیکل ماشین

مرجع سریع دستورات انتقال اطلاعات

ACALL adr11
LCALL adr16
RET
RETI
AJMP adr11
LJMP adr16
SJMP rel
JMP @A+DPTR

JZ rel
JNZ rel
CJNE A, direct, rel
CJNE A, #data, rel
CJNE Rn, #data , rel
CJNE @Ri, #data, rel
DJNZ Rn, rel
DJNZ direct, rel
NOP

راهنما

Rn	آدرس‌دهی ثبات R0 تا R7
direct	آدرس ۸ بیتی حافظه داده (RAM) داخلی (00H-FFH)
@Ri	آدرس‌دهی غیرمستقیم با استفاده از ثبات‌های R0 یا R1
source	بایت منبع که هر یک از ثبات‌های Rn، آدرس مستقیم (direct)، یا آدرس غیرمستقیم @Ri می‌تواند باشد.
dest	بایت مقصد که هر یک از ثبات‌های Rn، آدرس مستقیم (direct)، یا آدرس غیرمستقیم @Ri می‌تواند باشد.
#data	عدد ۸ بیتی در دستور
#data16	عدد ۱۶ بیتی در دستور
bit	آدرس ۸ بیتی یک بیت اطلاعات
rel	آدرس نسبی یا آفست ۸ بیتی علامت‌دار
addr11	آدرس ۱۱ بیتی برای صفحه 2k بایتی حافظه
addr16	