به نام خدا

تصحیح امتحان آخرترم ساختار و زبان کامپیوتر (۴۰۱۲۶) ۱ بهمن ۱۴۰۲

(مدت: ۲ ساعت) **از ۲۳ نمره**

۱- توضیح دهید (حداکثر در ۴ خط): - چرا حافظه نهان (Cache memory) در پردازنده‌های امروزی بسیار استفاده می‌شود؟ - چرا از دید برنامه‌نویس پنهان است؟ - چگونه کار می‌کند؟ - در چه شرایطی مفید واقع می‌شود؟ **(۲ نمره)**

چون می‌تواند سرعت دسترسی به حافظه را چند برابر سریعتر و به بهبود کارایی پردازنده یا سیستم رایانشی کمک کند؛ کنترلر حافظه نهان بر اساس سیاست جایابی و جایگزینی خود محتوای آدرس‌های مورد مراجعه اخیر را به گونه‌ای در حافظه نهان ذخیره می‌کند که احتمال استفاده مجدد آن‌ها در مراجعات بعدی بالا باشد (یعنی نرخ برخورد (Hit ratio) نزدیک به ۱)؛ در صورت برقراری اصل مجاورت مراجعات (اعم از مکانی، زمانی یا دنباله‌ای).

۲- هر چند انتظار می‌رود که کد اجرایی (مثلاً exe.) برنامه‌ها فقط حاصل ترجمه زبان سطح بالا یا اسمبلی به کد دودویی باشد ولی در عمل مشاهده می‌شود که گاه بعد از عمل Decompile یا Disassemble آن‌ها، برخی متغیرها و نماد (Symbol)های برنامه سطح بالا آشکار می‌شوند. ضمن مرور فرایند تولید کد باینری از زبان سطح بالا، دلیل این اتفاق و کاربردهای آن‌را توضیح دهید. **(۲ نمره)**

موقع ترجمه (Compile) برخی برنامه‌های سطح بالا، بسته به اینکه چه خط فرمانی را زده باشیم (یا کدام "سوئیچها" را فعال کرده باشیم)، کامپایلر برخی اطلاعات زبان سطح بالا مثل نام متغیرها و نمادها را بعد از Link برنامه نیز در بخشی از فایل باینری exe. ذخیره می‌کند. بنابراین برنامه‌های Decompiler یا Disassembler می‌توانند براحتی این اطلاعات را بازیابی کنند که هم در اشکال‌زدایی برنامه مفید است و هم در مهندسی معکوس آن.

۳- چرا امروزه پردازنده‌های RISC را به CISC ترجیح می‌دهند؟ شرکت‌هایی مثل اینتل چطور پردازنده‌های CISC خود (مثل 80x86) را با این روند تطبیق داده‌اند؟ **(۲ نمره)**

برای اینکه معماری RISC نوعاً سریعتر و ساده‌تر است همراه با تعداد ثبات زیاد و دستورات منظم. اینتل به صورت داخلی، دستورات سنتی CISC خود را به کمک یک سری دیکودر به RISC تبدیل می‌کند و با اینکه این موضوع هم تاخیر دارد و هم سخت‌افزار اضافی، ولی در مجموع، به کمک هسته RISC خود سریعتر از یک معماری تماماً CISC کار می‌کند.

۴- مدهای آدرس‌دهی پردازنده ARM را با 80x86 مقایسه کنید. در ضمن با یک مثال و دستورات مناسب نشان دهید که اگر ARM یک مد آدرس‌دهی مهم و معروف را همچون 80x86 ندارد، چگونه دسترسی به حافظه را با آن آدرس‌دهی پیاده کرده است. **(۳ نمره)**

مدهای آدرس‌دهی ARM:

Name	Alternative Name	ARM Examples
Register to register	Register direct	MOV R0, R1
Absolute*	Direct*	LDR R0, #Label
Literal	Immediate	MOV R0, #15 ADD R1, R2, #12
Indexed, base	Register indirect	LDR R0, [R1]
Pre-indexed,	Register indirect	LDR R0, [R1, #4]

base with displacement	with offset	
Pre-indexed, auto indexing	Register indirect pre-incrementing	LDR R0, [R1, #4]!
Post-indexing, auto indexed	Register indirect post-increment	LDR R0, [R1], #4
Double Reg indirect	Register indirect Register indexed	LDR R0, [R1, R2]
Double Reg indirect with scaling	Register indirect indexed with scaling	LDR R0, [R1, R2, LSL #2]
Program counter relative		LDR R0, [PC, #offset]

مدهای آدرس‌دهی 80x86:

Mode Name	Description or algorithm	Examples
Immediate	X	mov eax, 0x120
Register	R	mov eax, ebp
Register indirect	[R]	mov eax, [ebp]
Stack	[B]	pop ecx
Based	[B + disp]	mov eax, 0x2000[ebx]
Indexed	[I*scale + disp]	mov eax, 0x14[2*edi]
Based without scaled index and displacement	[B + I + disp]	mov eax, 0x14[ebx + 2*edi]
Based with scaled index and displacement	[B + I*scale + disp]	mov eax, 0x14[ebx + 2*edi]
Relative	[PC + disp]	jmp 0x120

[] = contents of ...
B = Base registers
R = General purpose registers
I = Index registers
PC = Program counter register
Scale = Scale factor = 1, 2, 4, 8
disp = displacement offset
X = Immediate value

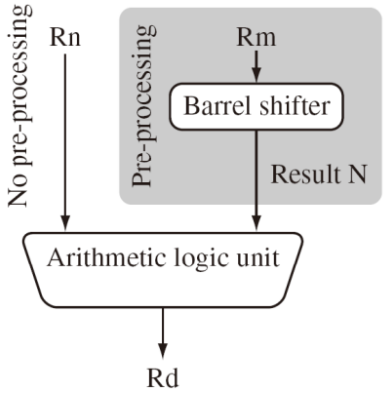
TABLE II. X86 SUPPORTED ADDRESSING MODE

* آدرس‌دهی مستقیم به یک خانه حافظه در ARM برخلاف 80x86 بر خلاف آنچه علی‌الظاهر در جدول آدرس‌دهی‌ها آمده است، امکان‌پذیر نیست و مستلزم دو دستور به شکل زیر است:

```
LDR r1, =x
LDR r2, [r1]
```

یعنی اول باید آدرس در یک ثبات ریخته شود و سپس از طریق آن خوانده شود.
(توجه شود که دستور mov برای عملوندهای ۱۶ بیتی است و برای بالاتر باید از LDR استفاده کرد)

۵- در پردازنده ARM امکان اجرای برخی دستورات حسابی توام با جابجایی عملوندها (Operands) به دلیل معماری زیر وجود دارد. (Barrel shifter یک جابجا کننده ورودی به تعداد بیت دلخواه است)



۵-۱ به طور مثال و با توجه به معماری فوق، توضیح دهید دستور زیر چگونه اجرا می‌شود و اگر R0=0xFA340056 و R2=0xA0000030 باشد، بعد از اجرای دستور زیر هر کدام چه مقداری خواهند داشت؟ (ASR یعنی Arithmetic shift right) **(۲ نمره)**

```
MOV  R0, R2, ASR #2 @ R0:=R2>>2
                    @ R2 unchanged
```

اول ثبات R2 به صورت حسابی (Arithmetic) یعنی با حفظ علامت، ۲ بیت به سمت راست جابجا و سپس در R2 ریخته می‌شود. از آنجا که R2=0xA0000030 و منفی است، پس از دو بار جابجایی حسابی به راست می‌شود:

R0=0xE800000C
R2=0xA0000030

۵-۲ جالب‌تر اینکه می‌توان جابجایی را به اندازه مشخص شده در یک ثبات (در مثال زیر: R3) روی یک ثبات (در اینجا: R2) و سپس عمل حسابی را انجام داد: **(۱ نمره)**

```
ADD  R0, R1, R2, LSL R3
```

مشخص کنید مقدار نهایی R0 به ازای سایر ثبات‌ها برابر چیست. (LSL: Logical shift left)

پاسخ:

$R0:=R1+R2*2^{R3}$

۶- نام انشعاب شرطی «بزرگتر از» را بعد از مقایسه اعداد علامت‌دار و شرط پرش برای هر پردازنده دلخواه مشخص کنید. **(۲ نمره)**

برای پردازنده ARM: BGT، (Z==0) && (S==V) پرچم S مربوط به بیت علامت و پرچم V مربوط به بیت سرریز و پرچم Z مربوط به صفرشدن خروجی عمل محاسبه یا مقایسه قبلی است.

برای پردازنده 80x86: JG، (Z==0) && (S==V)

۷- برنامه زیر به روش اِسنادی (یا همان دستورات مشروط) پیاده شده است:

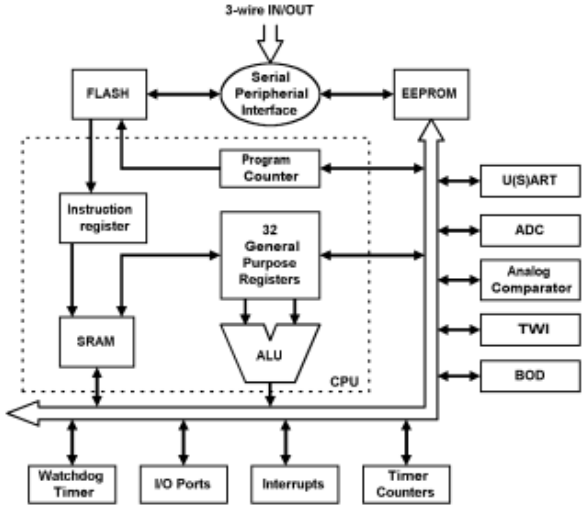
```
        CMP R0,R1
        CMPEQ R2,R3
        ADDEQ R4, R4, #1
Skip:   ...
```

مشخص کنید معادل کدام برنامه است یا چکار می‌کند. **(۲ نمره)**

پسوند EQ در دستورات دوم و سوم نشانگر اجرای شرطی این دو دستور به ازای شرط Z=1 یا برابری مقایسه است. بنابراین اگر در دستور اول R0=R1 بوده باشد یک مقایسه دیگر بین R2 و R3 صورت می‌گیرد و چنانچه این دو برابر باشند به R4 یکی اضافه می‌شود وگرنه پرش به ادامه برنامه رخ می‌دهد:

```
if (R0==R1)
 {if R2==R3
   R4= R4+1;
 }
... skip
```

۸- معماری داخلی یک میکروکنترلر AVR در زیر آمده است. (راهنما: یک میکروکنترلر شامل دستکم یک پردازنده و چندین درگاه یا رابط آنالوگ و دیجیتال با محیط پیرامونی و تعدادی حافظه است که به دلیل «جمع و جور» و ارزان بودن، در کاربردهای بسیاری مورد استفاده قرار گرفته یا می‌گیرد؛ TWI: Two-wire interface؛ زمان‌سنج مراقب :Watchdog timer؛ مدار کنترل ولتاژ :BOD: Brown out Detection؛ مبدل آنالوگ به دیجیتال :ADC؛ رابط سریال دو طرفه:USART)



۸-۱ اجزای مهم تشکیل دهنده آن و شیوه اجرای الگوریتم فون نیومن را به کمک عناصر شکل شرح دهید. **(۲ نمره)**

از شکل پیداست که ۳۲ ثبات همه‌منظوره داریم (که طبعاً با ۵ بیت قابل شماره‌گذاری و آدرس‌دهی است) و حافظه Flash برنامه اصلی را در خود ذخیره می‌کند و وقتی به ترتیب توسط PC مورد واکشی (Fetch) قرار می‌گیرد می‌تواند داده‌ها را در ثبات‌ها یا حافظه SRAM قرار دهد و یا از آن بخواند. این حافظه با قطع برق اطلاعاتش پاک می‌شود ولی حافظه Flash اطلاعات و برنامه را نگه می‌دارد و دفعه بعد که دستگاه را روشن کردید از همان دوباره برنامه پیش‌فرض (مثل کنترل ماشین رختشویی) را می‌خواند.

مدارات واسطه برای ارتباط با جهان خارج شامل:

- مدار واسطه دوسیمه (TWI: Two-wire interface) برای ارتباط سریال دوسیمه با دستگاه‌های مشابه. این ارتباط بسیار شبیه استاندارد I2C است که نرخ انتقال اطلاعاتی به صورت سنکرون برابر 100 kHz یا 400 kHz یا 3.4 MHz دارد. از ویژگی‌های استاندارد این است که چندین دستگاه می‌توانند خود را متصل به این دو سیم کنند (مثل لامپ‌هایی که با دو سیم به هم وصل هستند) و یکی نقش ارباب (Master) را بازی کند و بقیه به عنوان پیرو (Slave) با وی ارتباط برقرار کنند (در TWI می‌تواند تغییر کند ولی در I2C یکی است)
- زمان‌سنج مراقب (Watchdog timer) برای تنظیم زمان و برنامه‌ریزی انجام کارها مطابق یک حد استانه زمانی تا اگر از آن گذشت، پردازنده اقدامی کند یا فرمانی صادر کند (مثلاً اگر ظرف ۳۰ ثانیه دکمه‌ای را فشار ندادید، مداری را قطع کند).
- مدار کنترل ولتاژ (BOD: Brown out Detection) برای اینکه اگر ولتاز از یک حدی کمتر شد (مثلاً باتری در شرف اتمام بود، مدار را به یک وضعیت ایمن و معین ببرد.
- مبدل آنالوگ به دیجیتال(ADC) برای مقادیر سیگنال آنالوگ به دیجیتال جهت محاسبه توسط پردازنده

- رابط سریال دو طرفه(USART) برای ارتباط سریال با سایر دستگاه‌های سریال به صورت آسنکرون و بر خلاف TWI یا I2C نقطه به نقطه است.
- مدارات شمارنده زمانسنج (Timer counters) شمارش رویدادها یا امکان زمان‌بندی رویدادهای گوناگون را در اختیار این میکروکنترلر قرار می‌دهد.

الگوریتم فون نیومن با اشاره PC به حافظه Flash برای خواندن دستورالعمل شروع و PC بروزرسانی و سپس عمل کدگشایی (Decode) دستورالعمل برای تشخیص کار مورد نظر جهت اجرا توسط ALU یا انتقال از/به حافظه یا ورودی/خروجی‌ها انجام می‌شود ، (البته Decoder در شکل نشان داده نشده است). سپس عمل خواندن عملوندها از حافظه (چه SRAM و چه Flash و چه ثبات‌ها) صورت می‌گیرد و بعد از آن، دستورالعمل اجرا می‌شود. در این پردازنده، دستورالعمل می‌تواند یک یا دو عملوند را مورد ارجاع و استفاده قرار دهد. سپس عمل پس‌نویسی (Writeback) نتایج در حافظه (ثبات و یا SRAM) انجام می‌شود و این چرخه پس از پایان اجرای دستورالعمل یا بروز وقفه، از نو تکرار می‌شود.

۸-۲ با توجه به دو آدرسه و ۱۶ بیتی بودن دستورات، حداکثر چند دستور حسابی یا منطقی با دو ثبات می‌توان داشت؟ **(۱ نمره)**

از آنجا که دستورات و ثبات‌ها ۱۶ بیتی است، در حالتی که از دو ثبات در دستورالعمل استفاده کنیم، حداکثر ۶ بیت برای تعیین Opcode باقی می‌ماند و لذا حداکثر ۶۴ دستورالعمل دو آدرسه خواهیم داشت. (البته با توجه به اینکه دستورات فقط به این نوع آدرس‌دهی دو عملوندی محدود نمی‌شود و مثلاً دستورات تک آدرسه هم داریم، طبعاً تعداد کل دستورات می‌تواند بیش از مقدار فوق باشد ولی به هر حال، تعداد دستورات دو آدرسه نمی‌تواند بیش از ۶۴ تا باشد).

۸-۳ با توجه به جدول زیر، طرز پرش و/یا فراخوانی عادی، نسبی و غیر مستقیم را شرح دهید و با آنچه از دستورات مشابه 80x86 می‌شناسید مقایسه کنید. **(۳ نمره)**

Mnemonic	Operands	Description		Op	
RJMP	k	Relative Jump	PC	←	PC + k + 1
IJMP		Indirect Jump to (Z)	PC(15:0) PC(21:16)	← ←	Z 0
JMP	k	Jump	PC	←	k
RCALL	k	Relative Call Subroutine	PC	←	PC + k + 1
ICALL		Indirect Call to (Z)	PC(15:0) PC(21:16)	← ←	Z 0
EICALL		Extended Indirect Call to (Z)	PC(15:0) PC(21:16)	← ←	Z EIND
CALL	k	Call Subroutine	PC	←	k

منبع: AVR Instruction Set Manual (Atmel-0856L-AVR-Instruction-Set-Manual_Other-11/2016) **از شرکت Atmel**

قابل دریافت از پیوندهای زیر:

https://www.studocu.com/row/document/university-of-engineering-and-technology-peshawar/electronic-devices-and-circuits/atmel-0856-avr-instruction-set-manual/26858741

https://www.manualslib.com/manual/1402552/Atmel-Avr.html

https://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf

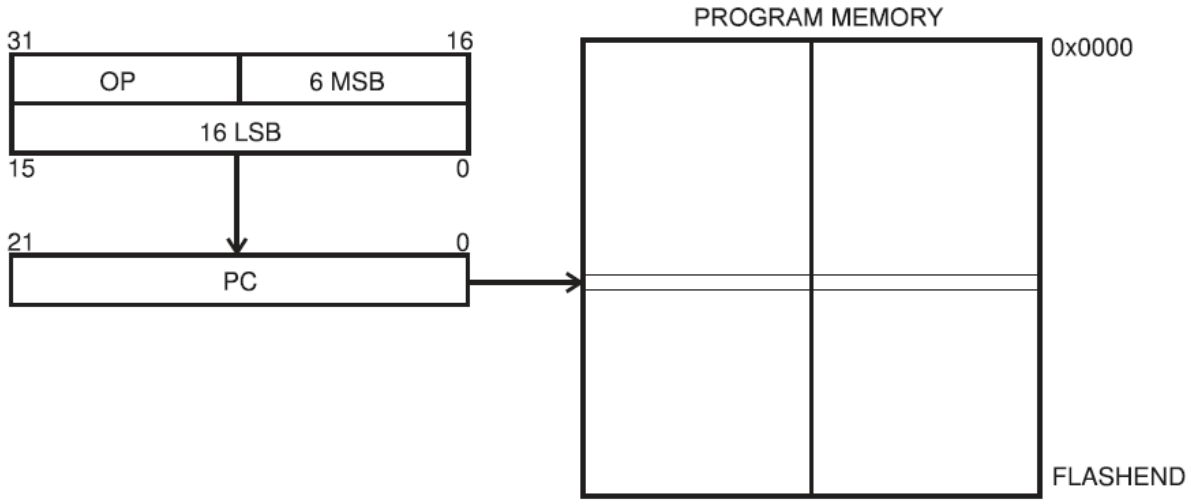
توجه کنید که در این پردازنده ثبات‌های R0 تا R31 ۸ بیتی‌اند و ثبات Z یک ثبات ۱۶ بیتی است که از الصاق دو ثبات R30 و R31 ایجاد و به عنوان اشاره‌گر در دستورات استفاده می‌شود.

X,Y,Z: Indirect Address Register (X=R27:R26, Y=R29:R28, and Z=R31:R30)

دستورات JMP و CALL ساده از نوع آدرس‌دهی مستقیم هستند که آدرس پرش در ادامه دستورالعمل تصریح و در PC جهت پرش ریخته می‌شود:

Direct Program Addressing, JMP and CALL

Figure 2-11. Direct Program Memory Addressing



Program execution continues at the address immediate in the instruction word.

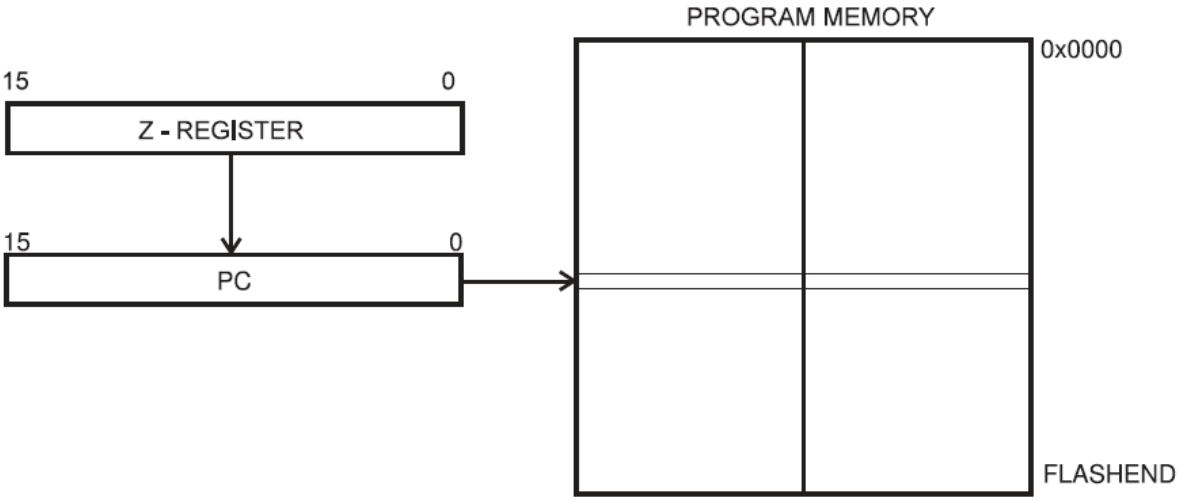
مثالی از برنامه حاوی دستور Call:

```
 mov r16,r0 ; Copy r0 to r16
 call check ; Call subroutine
 nop ; Continue (do nothing)
 ...
check: cpi r16,$42 ; Check if r16 has a special value
 breq error ; Branch if equal
 ret ; Return from subroutine
 ...
error: rjmp error ; Infinite loop
```

دستورات IJMP و ICALL از نوع آدرس‌دهی غیر مستقیم بر اساس ثبات اشاره‌گر Z هستند که آدرس پرش از Z داخل PC برای پرش ریخته می‌شود:

Indirect Program Addressing, IJMP and ICALL

Figure 2-12. Indirect Program Memory Addressing



Program execution continues at address contained by the Z-register (i.e., the PC is loaded with the contents of the Z-register).

مثالی از برنامه حاوی دستور IJMP:

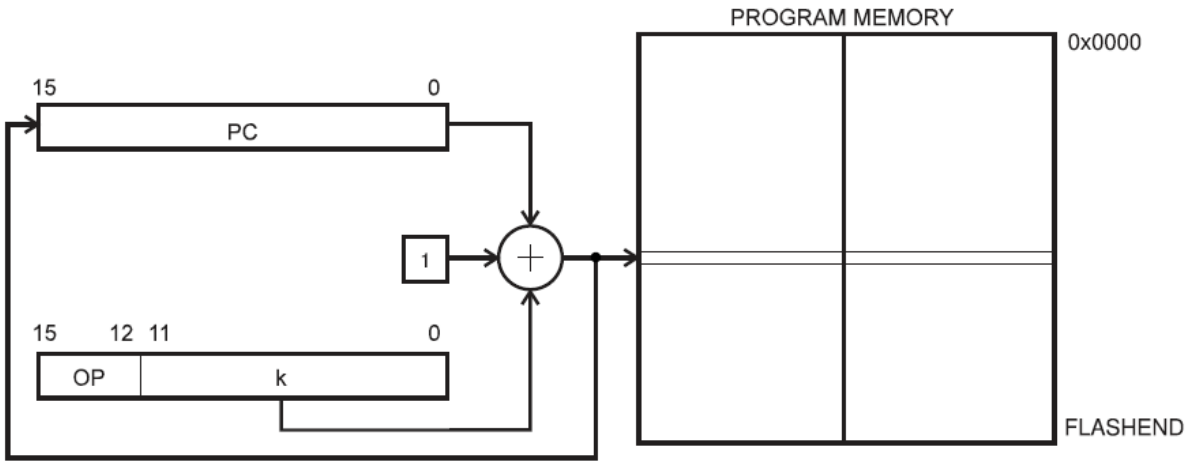
```
mov r30,r0 ; Set offset to jump table
ijmp ; Jump to routine pointed to by r31:r30
```

دستورات RJMP و RCALL از نوع آدرس‌دهی نسبی است که بر اساس مقدار جابجایی نسبت به PC (که فاصله با محل پرش، k، به صورت مکمل ۲ و ۱۲ بیتی، هنگام اسمبل کردن داخل دستورالعمل قرار می‌گیرد) محل پرش را محاسبه و داخل PC برای پرش می‌گذارد .

توجه کنید که دستورالعمل‌ها ۱۶ بیتی‌اند و کلمات حافظه Flash به صورت Word چیده شده است و مورد دسترسی قرار می‌گیرد. بنابراین وقتی PC یکی اضافه می‌شود، به دستور بعدی (نه بایت بعدی) اشاره می‌کند و به همین دلیل در پرش‌های نسبی، مقدار +1 را به k و PC اضافه می‌کنند.

Relative Program Addressing, RJMP and RCALL

Figure 2-13. Relative Program Memory Addressing



Program execution continues at address PC + k + 1. The relative address k is from -2048 to 2047.

دستور برای EICALL پرش به یک زیربرنامه «دوردست» (مشابه 80x86 Call far) است که ۶ بیت بالای ثبات EIND را با ۱۶ بیت ثبات Z ترکیب می‌کند و با این آدرس ۲۲ بیتی به سراغ اولین دستور زیربرنامه دوردست می‌رود:

EICALL – Extended Indirect Call to Subroutine

Description

Indirect call of a subroutine pointed to by the Z (16 bits) Pointer Register in the Register File and the EIND Register in the I/O space. This instruction allows for indirect calls to the entire 4M (words) Program memory space. See also ICALL. The Stack Pointer uses a post-decrement scheme during EICALL.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) PC(15:0) ← Z(15:0)
PC(21:16) ← EIND

	Syntax:	Operands:	Program Counter:	Stack:
(i)	EICALL	None	See Operation	STACK ← PC + 1 SP ← SP - 3 (3 bytes, 22 bits)

مثالی از یک برنامه با EICALL:

```
ldi r16,$05 ; Set up EIND and Z-pointer
out EIND,r16
ldi r30,$00
ldi r31,$10
eicall ; Call to $051000
```

در این برنامه، مقدار 05 را اول در R16 قرار داده، سپس در EIND و بعد به آدرس 051000 برای اجرای یک زیربرنامه پرش می‌کند.

دستورات پرش نسبی این پردازنده مشابه JMP Short و JMP near پردازنده 80x86 است با این تفاوت که در اینجا فاصله نسبی با یک عدد ۱۲ بیتی (نه ۸ برای حالت Short و نه ۱۶ بیت، برای حالت Near پردازنده‌های اینتل) کدگذاری می‌شود. حالت JMP و CALL مشابه آدرس دهی مستقیم و EIJMP و EICALL آدرس‌دهی غیر مستقیم بر اساس یک ثبات در پردازنده‌های اینتل است.

۸-۴ چه کاربردهایی را برای این پردازنده می‌توانید تصور کنید؟ **(۱ نمره)**

کاربردهای متنوع کنترلی و داده‌برداری و اینترنت اشیاء، سیستم‌های نهفته مثل کنترلر ماشین رختشویی و یخچال، کنترل روشنایی، دما، ابیاری خودکار گیاهان و خانه هوشمند و مشابه قابل ذکر است.