بسمه تعالی

پاسخنامه تکلیف اول درس ریزپردازنده

استاد درس:

دکتر حمیدرضا حکیم داوودی

دستیار آموزشی:

مسیح تنورساز

موعد تحویل

چهاردهم آبان

نیمسال اول 1403-1404

توجه نمایید پاسخ سوالات با این رنگ مشخص شده اند.
راه‌حل‌های ارائه شده در این پاسخنامه تنها راه‌حل‌های موجود برای حل سوالات 4، 5، 7 و 8 نمی‌باشند. هرگونه راه‌حل درست نمره‌ی سوال را دریافت خواهد کرد.

---

1) تفاوت دو معماری وان نیومن و هاروارد را بیان کنید و چند مورد از مزیت‌های هرکدام را نام ببرید.(5 نمره)

پاسخ:

1. معماری وان نیومن:

در معماری وان نیومن، **داده‌ها** و **دستورات** در یک حافظه مشترک ذخیره می‌شوند و پردازنده از طریق یک باس واحد به آن‌ها دسترسی دارد.

- **مزایا:**
  - **سادگی طراحی**: ساختار ساده‌تر و راحت‌تر برای پیاده‌سازی.
- **معایب:**
  - **مشکل گلوگاه وان نیومن**: به دلیل استفاده از یک باس برای انتقال داده‌ها و دستورات، سرعت پردازش محدودتر است.

2. معماری هاروارد:

در معماری هاروارد، **حافظه داده‌ها** و **حافظه دستورات** از هم جدا هستند و پردازنده می‌تواند از دو باس مجزا برای دسترسی به آن‌ها استفاده کند.

- **مزایا:**
  - **افزایش کارایی**: پردازنده می‌تواند به طور همزمان به داده‌ها و دستورات دسترسی پیدا کند و این امر به بهبود عملکرد منجر می‌شود. همچنین دقت نمایید که جدا بودن حافظه‌ها می‌تواند از تغییرات ناخواسته در دستورات جلوگیری کند.
- **معایب:**
  - **پیچیدگی بیشتر**: طراحی پیچیده‌تر و هزینه بالاتر برای تولید.

2) با توجه به تفاوت‌های بین Microcontroller و Microprocessor، به سوالات زیر پاسخ دهید:(5 نمره)

- تفاوت‌های اصلی بین معماری میکروکنترلر و میکروپروسسور چیست؟
- برای هر یک از میکروکنترلر و میکروپروسسور، یک کاربرد عملی را بیان کرده و توضیح دهید چرا یکی از این دو برای آن کاربرد مناسب‌تر است.
- به طور معمول، کدام یک از این دو معماری مصرف انرژی و هزینه کمتری دارند و در چه نوع کاربردهایی این تفاوت‌ها اهمیت پیدا می‌کنند؟

پاسخ:

- میکروکنترلر: یک سیستم کامل روی یک تراشه است که شامل پردازنده، حافظه (RAM و ROM یا Flash)، تایمرها، پورت‌های ورودی/خروجی (I/O) و سایر ماژول‌های جانبی (مانند ADC) می‌باشد.
- میکروپروسسور: تنها یک واحد پردازش مرکزی (CPU) است که برای عملکرد نیاز به قطعات جانبی مانند ROM، RAM و I/Oهای خارجی دارد.

کاربرد عملی برای هر یک:

- میکروکنترلر:
  - استفاده در ماشین لباسشویی، میکروکنترلرها برای این کاربرد مناسب‌تر هستند زیرا تمام اجزای مورد نیاز برای کنترل فرآیند در یک تراشه واحد قرار دارند، که منجر به مصرف انرژی پایین‌تر و کاهش هزینه می‌شود.

- میکروپروسسور:
  - استفاده در رایانه‌ها و لپ‌تاپ‌ها، میکروپروسسورها برای این کاربرد مناسب‌تر هستند زیرا توانایی پردازش داده‌های پیچیده را با سرعت بالا دارند.

مصرف انرژی و هزینه:

- میکروکنترلرها به طور معمول مصرف انرژی و هزینه کمتری دارند زیرا برای کارهای ساده‌تر و کاربردهای تعبیه‌شده طراحی شده‌اند.
- میکروپروسسورها معمولاً انرژی بیشتری مصرف می‌کنند و هزینه بالاتری دارند، زیرا برای پردازش محاسبات سنگین و پیچیده طراحی شده‌اند.

کاربردهایی که تفاوت‌ها اهمیت پیدا می‌کنند:

- میکروکنترلرها در کاربردهایی مانند دستگاه‌های اینترنت اشیاء (IoT)، دستگاه‌های قابل حمل (مانند ساعت‌های هوشمند) و ابزارهای صنعتی کوچک اهمیت دارند، چرا که مصرف انرژی پایین و هزینه کمتر در این موارد کلیدی است.
- میکروپروسسورها در کاربردهایی مثل سرورها، کامپیوترهای رومیزی و در حالت کلی، جایی که قدرت پردازشی و سرعت بیشتر اهمیت پیدا می‌کند استفاده می‌شوند.

تراشه مورد استفاده در این درس که ATmega32 خواهد بود نیز یک میکروکنترلر می‌باشد.

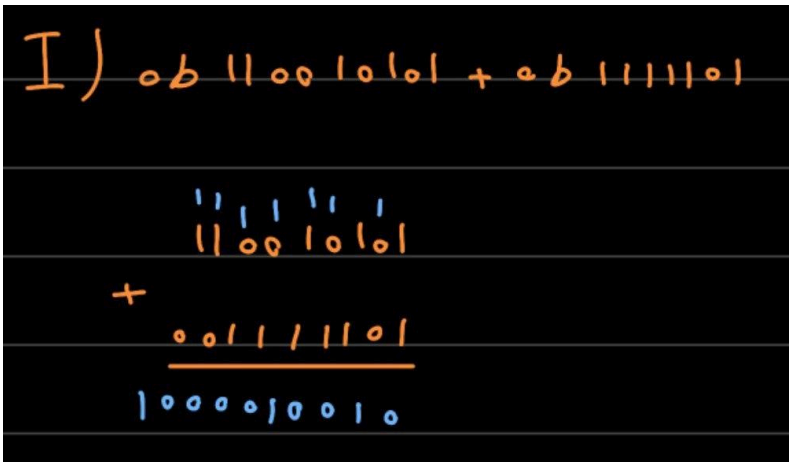
3) جدول زیر را تکمیل نمایید. (16 نمره)

Decimal	Binary	Hexadecimal
1,277	1101 1111 100	4FD
3,889	111100110001	F31
35,421	1101 0101 1010 1000	8A5D
2024	1000 1110 111	7E8

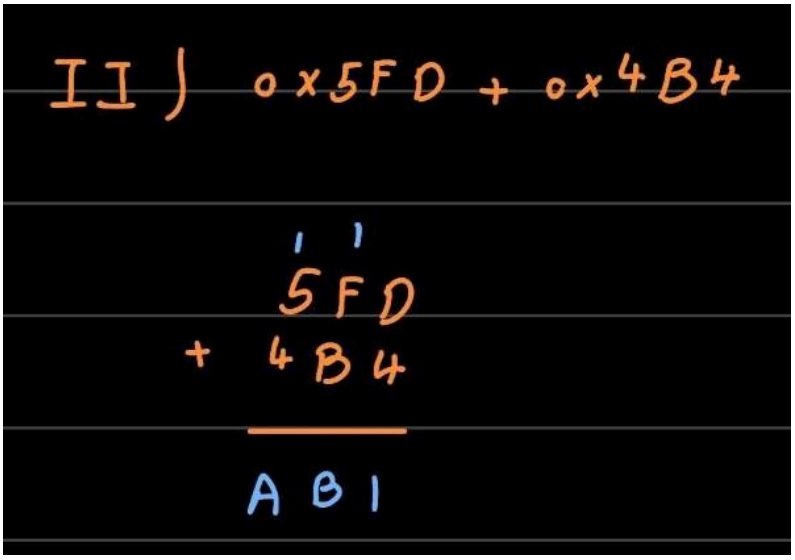
4) عبارت‌های زیر را محاسبه نمایید.(ارائه راه حل کامل الزامی است.) (30 نمره)

پاسخ:

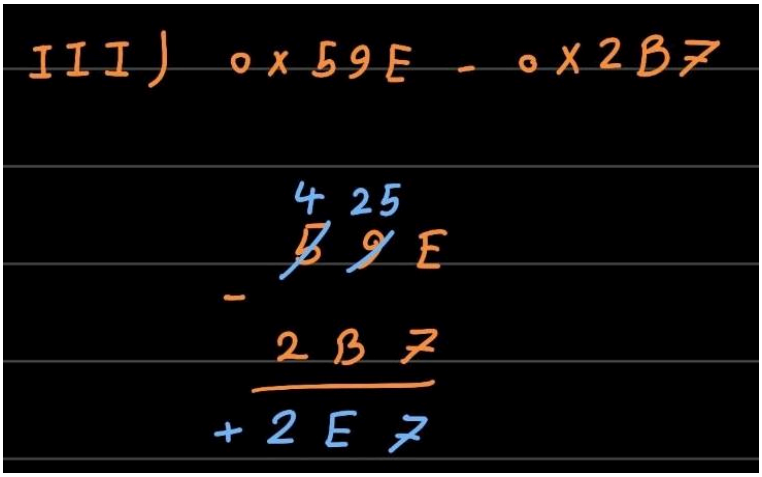
I. 0b110010101 + 0b1111101



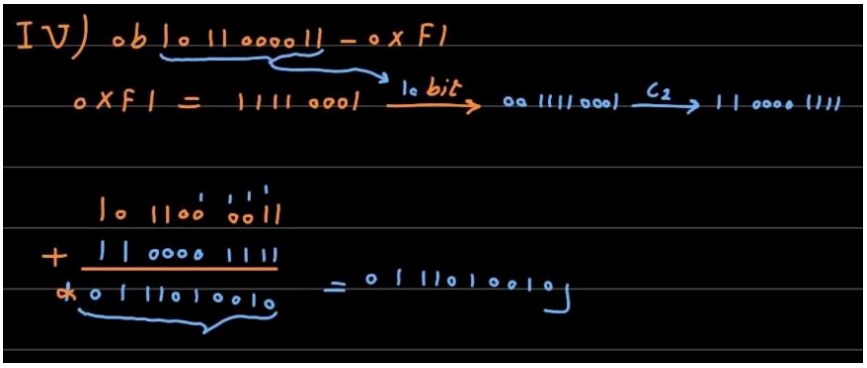
II. 0x5FD + 0x4B4



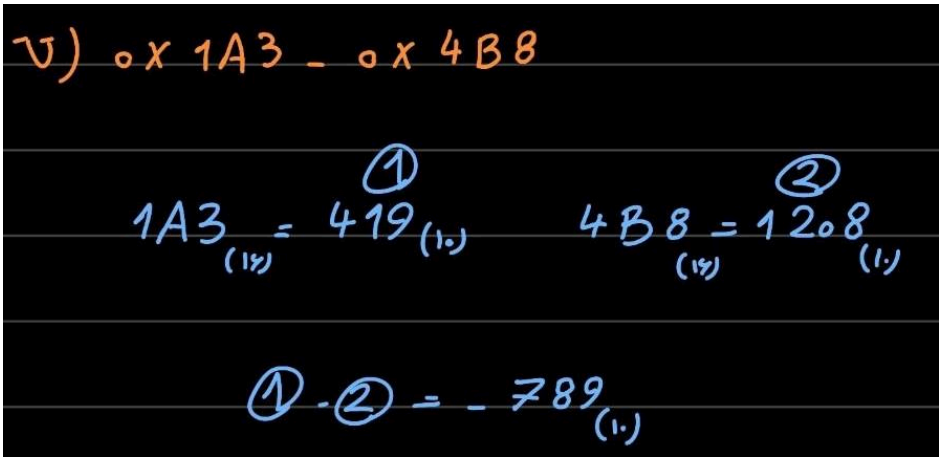
III. 0x59E - 0x2B7



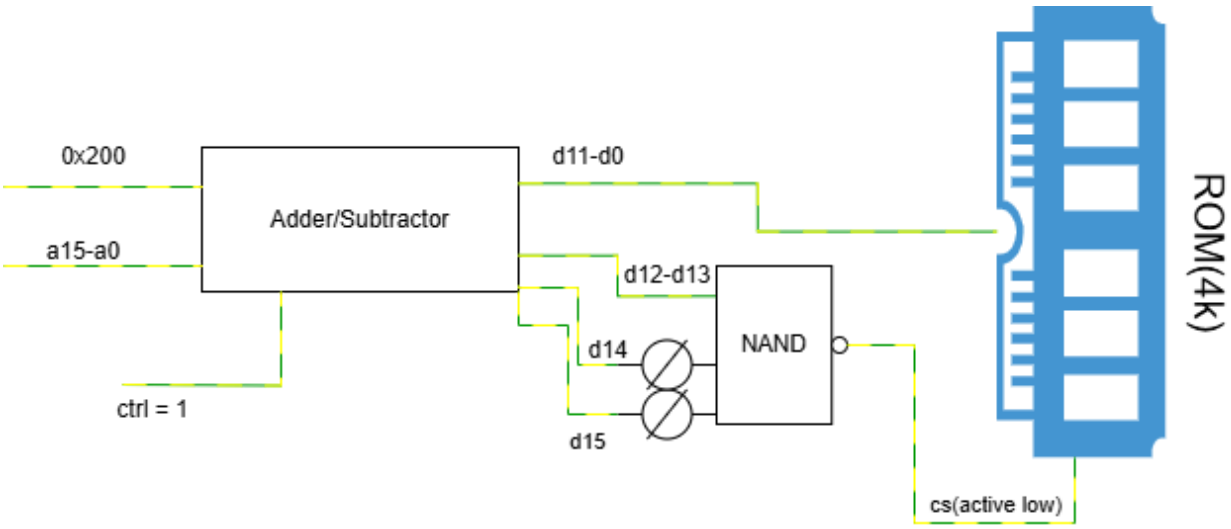
IV. 0b1011000011 – 0xF1 (به روش مکمل دو حل شود)



V. 0x1A3 - 0x4B8 (به روش دلخواه حل شود)



5) مدار مرتبط با mapکردن یک حافظه ROM با capacity 4k، از 3200H تا 41FF را رسم کنید.(راهنمایی: میتوانید از Half-Adder استفاده کنید.) (34 نمره)



برای حل این سوال می‌توان 0x200 را از بازه‌ی داده شده کم کرد، با انجام این کار بازه به صورت زیر تعریف می‌شود:

0x3200 - 0x200 = 0x3000 ⟶ 0b0011 0000 0000 0000

0x41FF - 0x200 = 0x3FFF ⟶ 0b0011 1111 1111 1111

همانطور که مشاهده می‌شود 4 بیت پر ارزش ابتدا و انتهای بازه یکسان شده است(هردو 0b0011 شده است). این 4 بیت را به چیپ سلکت مموری متصل خواهیم کرد و 12 بیت کم ارزش خطوط آدرس را تشکیل می‌دهند.

6) معماری RISC و CISC را با یکدیگر مقایسه کنید و توضیح دهید که شما کدام یک را مناسب‌تر می‌بینید. (10 نمره)

پاسخ:

معماری‌های RISC و CISC دو رویکرد متفاوت در طراحی CPU هستند.

مقایسه دو معماری:

- RISC تعداد دستورالعمل‌های محدودی دارد که هر یک ساده و بهینه شده‌اند. اجرای هر دستور معمولاً در یک سیکل ساعت انجام می‌شود که باعث سرعت بالاتر در پردازش می‌شود.
- CISC دارای مجموعه‌ای پیچیده و متنوع از دستورالعمل‌ها است که می‌توانند چندین عملیات را در یک دستور ترکیب کنند. این کار می‌تواند منجر به اجرای دستورها با سیکل‌های بیشتر شود، اما در عوض پیچیدگی نرم‌افزار کاهش می‌یابد.

کاربردها:

- معماری RISC بیشتر در پردازنده‌های موبایل، دستگاه‌های تعبیه‌شده و سیستم‌های که نیاز به توان مصرفی کم و سرعت بالا دارند، مانند سری پردازنده‌های AMD، استفاده می‌شود.
- معماری CISC در پردازنده‌های کامپیوترهای رومیزی و سرورها که نیاز به انعطاف‌پذیری و اجرای نرم‌افزارهای متنوع دارند، مانند سری پردازنده‌های Intel، استفاده می‌شود.

نتیجه نهایی:

انتخاب بین RISC و CISC بستگی به کاربرد مورد نظر دارد. در سیستم‌هایی که بهینه‌سازی مصرف انرژی و سرعت بالا اهمیت دارد، مانند موبایل‌ها و دستگاه‌های اینترنت اشیا، معماری RISC مناسب‌تر است. از سوی دیگر، برای سیستم‌های دسکتاپ و سرورها که نیاز به سازگاری نرم‌افزاری بالا و پشتیبانی از دستورالعمل‌های پیچیده دارند، معماری CISC مناسب‌تر است.

7) فرض کنید درون آدرس 0x315 حافظه مقدار 0xfd قرار دارد. برنامه‌ای به زبان اسمبلی بنویسید که مقدار 0xfd را به دسیمال تبدیل کند. ارقام عدد دسیمال تبدیل شده را در خانه های

0x322 ، 0x323 و 0x324 حافظه ذخیره کنید.(در خانه 0x322 رقم کم ارزش عدد دسیمال را

```
.EQU HEX_NUM = 0×315
.EQU RMND_L = 0×322
.EQU RMND_M = 0×323
.EQU RMND_H = 0×324

.DEF NUM = R20
.DEF DENOMINATOR = R21
.DEF QUOTIENT = R22

LDI R16, 0×FD          ; $FD = 253 in decimal
STS HEX_NUM, R16       ; store $FD in location 0×315

LDS NUM, HEX_NUM
LDI DENOMINATOR, 10    ; DENOMINATOR = 10

L1:
INC QUOTIENT
SUB NUM, DENOMINATOR
BRCC L1                ; if C = 0 go back

DEC QUOTIENT           ; once too many
ADD NUM, DENOMINATOR   ; add back to it
STS RMND_L, NUM        ; store remainder as the 1st
digit
MOV NUM, QUOTIENT
LDI QUOTIENT, 0

L2:
INC QUOTIENT
SUB NUM, DENOMINATOR
BRCC L2                ; if C = 0 go back

DEC QUOTIENT           ; once too many
ADD NUM, DENOMINATOR   ; add back to it
STS RMND_M, NUM        ; store remainder as the 2nd
digit
STS RMND_H, QUOTIENT   ; store quotient as the 3rd digit

HERE: JMPHER         ; stay here forever
```

قرار دهید.)(40 نمره)

8) برنامه ای به زبان اسمبلی بنویسید که فاکتوریل یک عدد صحیح را محاسبه کند. تضمین می‌شود عدد ورودی بین 0 تا 5 است. برنامه شما باید عدد ورودی از ادرس 0x400 حافظه بخواند. اگر عدد ورودی صفر بود تابع باید یک را برگرداند.(60 نمره)

(سعی کنید با تابع (Function) این کار را انجام دهید)

```
.DEF NUM = R16              ; Register for input number
.DEF FACTORIAL = R17        ; Register to store the factorial result
.DEF TEMP = R18             ; Temporary register for loop

.EQU INPUT_ADDR = 0x400     ; Memory address for input
.EQU OUTPUT_ADDR = 0x401    ; Memory address for output

MAIN_LOOP:
    LDS NUM, INPUT_ADDR     ; Load the input number from memory
    CPI NUM, 0              ; Compare the input number with 0
    BREQ RETURN_ONE         ; If the input is 0, branch to RETURN_ONE

    LDI FACTORIAL, 1        ; Initialize factorial to 1

FACTORIAL_LOOP:
    CP NUM, 1               ; Check if the number is 1
    BRLE END_LOOP           ; If number is less or equal to 1, exit loop

    MOV TEMP, NUM           ; Copy NUM to TEMP
    MUL FACTORIAL, TEMP     ; Multiply FACTORIAL by TEMP
    MOV FACTORIAL, R0       ; Move the result to FACTORIAL (use only R0)
    CLR R1                  ; Clear R1 to ensure it is 0 (optional, better to do this)
    DEC NUM                 ; Decrease NUM by 1
    RJMP FACTORIAL_LOOP     ; Repeat the loop to the first state

END_LOOP:
    STS OUTPUT_ADDR, FACTORIAL ; Store the result in memory
    RJMP MAIN_LOOP             ; Jump back to main loop for next input

RETURN_ONE:
    LDI FACTORIAL, 1        ; Set factorial to 1 for input 0
    STS OUTPUT_ADDR, FACTORIAL ; Store the result in memory
    RJMP MAIN_LOOP          ; Jump back to main loop for next input
```

پاسخ نمونه سوالات میان‌ترم

1) یک هارد دیسک داریم که می‌تواند 2 گیگابایت اطلاعات را در خودش ذخیره کند. فرض کنید هر صفحه از متن 25 سطر دارد که هر سطر هم 80 ستون دارد که در هرکدام از خانه های این صفحه میتوان یک کاراکتر اسکی(ASCII) ذخیره کرد که هر کاراکتر یک بایت است. تقریبا چند صفحه از اطلاعات را میتوان با استفاده از این هارد ذخیره کرد؟

پاسخ:

1byte*80*25=2000

از آنجایی که مقدار تقریبی را می‌خواهیم می‌توانیم هر گیگابایت را به جای

1,073,741,824بایت، 10 به توان 9 بایت در نظر بگیریم و یا در انتها جواب را تقریبی

بنویسیم.

2*10^9byte / 2*10^3byte = 10^6 ≈ 1000000 page

و یا بطور دقیق‌تر

2*1073741824 / 2000 ≈ 1073742 page

2) کد اسمبلی بنویسید که رشته‌ای از کاراکترها را از پورت A دریافت کند و در صورتی که حروف دریافتی کوچک باشند آنها را به حروف بزرگ تبدیل کند.

پاسخ:

```
.EQU INPUT_PORT = PINA      ; Register for reading from Port A
.EQU OUTPUT_PORT = PORTB    ; Register for writing to Port B

.DEF CHAR = R16             ; Register to hold the current character

; Initialize ports
INIT_PORTS:
    LDI R20, 0x00           ; Set all bits of Port A as input
    OUT DDRA, R20           ; Write to DDRA to configure Port A as input
    LDI R20, 0xFF           ; Set all bits of Port B as output
    OUT DDRB, R20           ; Write to DDRB to configure Port B as output

; Main program loop
MAIN_LOOP:
    IN CHAR, INPUT_PORT     ; Read the character from Port A
    CPI CHAR, 'a'           ; Compare with 'a' (ASCII 97)
    BRLO OUTPUT_CHAR        ; If CHAR < 'a', skip to output (not a lowercase letter)
    CPI CHAR, 'z'           ; Compare with 'z' (ASCII 122)
    BRHI OUTPUT_CHAR        ; If CHAR > 'z', skip to output (not a lowercase letter)

    ; Convert lowercase to uppercase
    SUBI CHAR, 0x20         ; Subtract 32 (0x20 in hex) to convert to uppercase

OUTPUT_CHAR:
    OUT OUTPUT_PORT, CHAR   ; Output the character to Port B
    RJMP MAIN_LOOP          ; Repeat the loop indefinitely
```