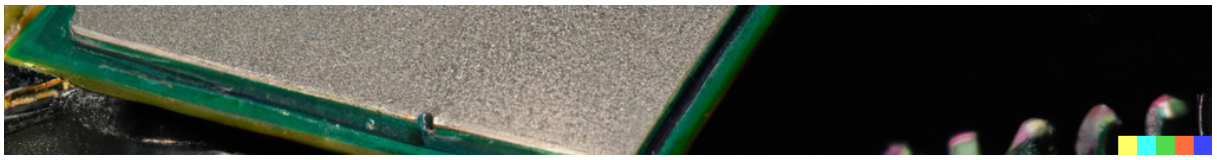
میپس!

شرک و خر وی، تصمیم گرفتند در دوره برنامه نویسی اسمبلی شرکت کنند تا پس از اخذ مدرک بتوانند کاری پیدا کنند و خرج زندگی‌شان را در بیاورند. در آزمون نهایی این دوره که به صورت آنلاین و از راه دور برگزار می‌شود، شرک خیلی سریع تمامی سوالات را فول کرد و نمره کامل دریافت کرد. اما خره نتوانسته سوالات را حل کند و از اینکه نمی‌تواند مدرک دوره را بگیرد و خرج بچه‌هایش را بدهد بسیار ناراحت است. شرک بعد از این که از این موضوع خبردار می‌شود سعی می‌کند تا جواب سوال‌های دوره را برای خره بفرستد تا او هم موفق به دریافت مدرکش شود. اما این وسط شرک متوجه می‌شود که جواب‌هاش را به صورت فایل باینری نگهداری کرده و کد اسمبلی را از دستگاه خود پاک کرده است و همچنین فرصت هم ندارد که دوباره از اول جواب سوالات را بنویسد. برای همین از شما که قادر به تبدیل کد اسمبلی به باینری (و برعکس) هستید، میخواهد تا فایل سوالات را به کد اسمبلی برگردانید و به او بدهید تا بتواند برای خره بفرستد و از این مخمصه نجاتش دهد.



*توضیحات: * پردازنده مورد استفاده شرک از معماری MIPS استفاده می‌کند. در این معماری ۳۲ رجیستر (حافظه) ۳۲ بیتی وجود دارد که در دستورات پردازنده تنها می‌توان از این رجیسترها استفاده کرد. همچنین در این معماری تمامی دستورات در ۳۲ بیت ذخیره می‌شوند و این دستورات نیز به سه دسته تقسیم می‌شوند.

۱. I - Format
۲. J - Format
۳. R - Format

در ادامه جدول دستوراتی که پردازنده شرک از آن‌ها پشتیبانی می‌کند آورده شده است، همچنین توضیحات بخش‌های مختلف دستورات نیز از جدول قابل استنتاج است.

جدول I Format

name	6 bits	5 bits	5 bits	16 bits	example
	operation code (hex)	rs	rt	immediate	
addi	08	1	2	10	addi $2, $1, 10
andi	0C	3	4	11	andi $4, $3, 11
ori	0D	5	6	12	ori $6, $5, 12
lw	23	7	8	13	lw $8, 13($7)
lb	20	9	10	14	lb $10, 14($9)
sw	2B	11	12	15	sw $12, 15($11)
sb	28	13	14	16	sb $14, 16($13)

جدول J Format

name	6 bits	26 bits	example
	op (hex)	address (hex)	
j	02	ABC2EE	j ABC2EE
jal	03	EF1212	jal EF1212

جدول R Format

name	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
	op (hex)	rs	rt	rd	shamt	func (hex)	example
sll	00	0	2	3	3	00	sll $3, $2, 3
srl	00	0	12	3	5	02	srl $3, $12, 5
add	00	1	2	3	0	20	add $3, $1, $2
sub	00	2	3	1	0	22	sub $1, $2, $3
or	00	2	3	4	0	25	or $4, $2, $3
xor	00	1	2	3	0	26	xor $3, $1, $2

- هر رجیستر به فرم زیر نمایش داده میشود:

```
$ + (register number)
```

برای مثال رجیستر شماره 4 به صورت 4$ نمایش داده میشود.

- همانطور که در جدول قابل مشاهده است، در تمامی دستورات ۶ بیت به op code (برای تمایز دستورات از یکدیگر) اختصاص یافته است.
- در دستورات I Format پنج بیت به رجیستر مبدا و پنج بیت به رجیستر مقصد اختصاص داده شده است، همچنین ۱۶ بیت برای مقدار immediate قرار دارد.
- در دستورات J Format علاوه بر opcode تنها یک بخش برای آدرس وجود دارد که دستورات این قسمت به جامپ (مشابه دستور goto عمل می‌کنند.) اختصاص دارند.
- دستورات R-Format شامل ۳ بخش برای ۳ رجیستر مقصد، عملوند اول و عملوند دوم است و همچنین یک بخش شامل ۵ بیت برای مقدار shift amount اختصاص دارد و همچنین ۶ بیت برای function code قرار داده شده است. احتمالا تا الان متوجه شده اید که دستورات R - format به جای opcode با مقدار function code از یکدیگر تمایز پیدا می‌کنند.

در ادامه چند مثال از دستورات جداول بالا می‌آوریم و سپس به فرایند تبدیل دستورات به کد باینری می‌پردازیم.

```
andi $4, $3, 11
```

- دستور فوق که جز دستورهای i format است، مقدار رجیستر ۴ را برابر حاصل جمع مقدار رجیستر ۳ و عدد ۱۱ می‌کند.

```
sw $12, 15($11)
```

- دستور بالا مقدار رجیستر ۱۲ را در خانه‌ای از حافظه که آدرسش برابر حاصل جمع رجیستر ۱۱ و عدد ۱۵ است ذخیره می‌کند.

```
j ABC2EE
```

- دستور فوق همانند دستور goto در زبان c عمل می‌کند و به جایی از حافظه می‌رود که آدرس آن برابر ABC2EE است.

```
sll $3, $2, 3
```

- با اجرای دستور فوق، مقدار رجیستر ۳ برابر حاصل ۳ بیت شیفت رجیستر ۲ به سمت چپ می‌شود.

برای تبدیل کد اسمبلی به باینری، می‌توانیم از جدول فوق استفاده کنیم، برای انجام این کار، می‌بایست بیت‌های باینری بخش‌های مختلف را از سمت چپ کنار یکدیگر قرار دهیم تا یک دستور ۳۲ بیتی تشکیل شود، برای مثال به سراغ کد کردن دستورهای زیر می‌رویم.

```
addi $2, $1, 10
```

این دستور جز دستورات I-Format است پس از این جدول استفاده می‌کنیم:

```
opcode: 001000
rs: 00001
rt: 00010
immediate: 0000000000001010
addi $2, $1, 10: 00100000001000100000000000001010, Hex: 2022000A
```

```
lw $8, 13($7)
opcode: 100011 (23 in HEX)
rs: 00111
```

```
rt: 01000
immediate: 0000000000001101
lw $8, 13($7) : 10001100111010000000000000001101, Hex: 8CE8000D


j ABC2EE
opcode: 000010
address: 00101010111100001011101110
j ABC2EE: 00001000101010111100001011101110, Hex: 08ABC2EE


sll $3, $2, 3
opcode: 000000
rs: 00000
rt: 00010
rd: 00011
shift amount: 00011
function code: 000000
sll $3, $2, 3 : 00000000000000100001100011000000, Hex: 000218C0


xor $3, $1, $2
opcode: 000000
rs: 00001
rt: 00010
rd: 00011
shift amount: 00000
function code: 100110
xor $3, $1, $2: 00000000001000100001100000100110, Hex: 00221826
```

شرک امیدوار است که تا اینجای سوال را به خوبی مطالعه کرده باشید! زیرا او نیاز دارد تا شما برعکس این کار را انجام بدهید، یعنی فایل حاوی ۰ و اهای برنامه شرک را گرفته و دستورات اسمبلی متناظر با آن را درست کنید.

نکته: توجه کنید که ورودی برنامه به شکل بایت ذخیره شده است و نه متن!

ورودی

هنگام اجرای برنامه، نام یک فایل باینری به عنوان آرگومان کامند لاین (مشابه زیر) به برنامه شما داده می‌شود که آن فایل در کنار کد شما قرار دارد.

```
./output filename.dat
```

برای سادگی می‌توانید از کد زیر برای ورودی گرفتن نام فایل استفاده کنید:

```c
#include <stdio.h>

// Your functions:

int main(int argc, char** argv){
    if(argc != 2){
        printf("Usage: %s <filename>\n", argv[0]);
        return 1;
    }
    FILE* file = fopen(argv[1], "rb");
    if(file == NULL){
        printf("File not found\n");
        return 1;
    }
    // Your Code:

}
```

خروجی

در خروجی باید دستورات متناظر با فایل باینری را چاپ کنید.

مثال

ورودی نمونه

```
20 22 00 0A 8c E8 00 0D 08 AB C2 EE 00 02 18 C0
(این بایت‌ها در یک فایل ذخیره شده است.)
```

خروجی نمونه

```
addi $2, $1, 10
lw $8, 13($7)
```

```
j ABC2EE
sll $3, $2, 3
```

داوری

با توجه به اینکه این سوال داوری خودکار ندارد، داوری لوکال آن در اینجا قرار گرفته است و می‌توانید از آن برای صحت کارکرد کد استفاده کنید.