به نام خداوند بخشنده مهربان

معماری کامپیوتر

دانشکده مهندسی کامپیوتر

دکتر اسدی
بهار ۱۴۰۳

مهدی علی نژاد، ۴۰۱۱۰۶۲۶۶

تمرین دهم

سوال ۱

۱. فرض کنید ماشینی با دو حالت آدرس‌دهی داریم که ویژگی‌های زیر را دارد:

۱. حافظه اصلی: با اندازه ۶۴ کلمه که هر کلمه آن ۸ بیت است.
۲. حافظه نهان: با اندازه ۱۶ کلمه که هر بلاک حافظه نهان ۸ کلمه دارد و سیاست آن نگاشت مستقیم[۱] است.
۳. ۴ ثبات عمومی[۲] با نام‌های R0, R1, R2, R3 دارد.
۴. ذخیره‌سازی اعداد به‌صورت مکمل ۲[۳] است.
۵. پشتیبانی از حالات آدرس‌دهی مستقیم[۴] و فوری[۵] را دارد.

دو مدل دستور زیر را درنظر بگیرید:

Opcode	Address
2 bits	6 bits

Opcode	register	Opr2
3 bits	2 bits	3 bits

و فرض کنید دستورات ماشین ما به صورت زیر است:

Opcode	Mnemonic	Operation
01	jmp address	PC <- Address
10	jnz address	if (R0) != 0 then PC <- Address
11	jz address	if (R0) = 0 then PC <- Address
000	add r1,data	r1 <- (r1) + data; t =1
000	add r1,r2	r1 <- (r1) + (r2); t =0
001	sub r1,data	r1 <- (r1) - data; t =1
001	sub r1,r2	r1 <- (r1) - (r2); t = 0

در دستورات add ردیف چهارم و sub ردیف ششم، عملگر سوم ۳ بیتی است که اگر پرارزش‌ترین بیت آن (t) برابر ۰ باشد یعنی یک ثبات مدنظر است و اگر ۱ باشد یعنی یک داده فوری[۶] است. حال با توجه به ماشین داده شده به سوالات زیر پاسخ دهید:

آ) مطابق ساختار حافظه نهان در اسلایدها، ساختار حافظه نهان این کامپیوتر را رسم کنید و مشخص کنید که هر قسمت چند بیت است.

ب) نرخ برخورد[۷] برنامه زیر را حساب کنید (شروع برنامه از آدرس 14h حافظه اصلی است)
20h, 26h, 0, 0, 0, 5Eh, 2Eh, 28h, BFh, BFh, 25h, 99h, 0

آ)

ISA		
tag(2bit)	set index(1bit)	offset (3bit)

cache			
set	tag	data	valid bit
0	(2 bit)	(8 byte)	(1 bit)
1	(2 bit)	(8 byte)	(1 bit)

(ب) برنامه به این صورت است:

address	hex inst.	binary instruction	instruction
14h	20h	00100000	sub r0, r0
15h	26h	00100110	sub r0, -2
16h	00h	00000000	add r0, r0
17h	00h	00000000	add r0, r0
18h	00h	00000000	add r0, r0
19h	5Eh	01011110	jmp 1Eh
1Ah	2Eh	00101110	sub r1, -2
1Bh	28h	00101000	sub r1, r0
1Ch	BFh	10111111	jnz 3Fh
1Dh	BFh	10111111	jnz 3Fh
1Eh	25h	00100101	sub r0, 1
1Fh	99h	10011001	jnz 19h
20h	00h	00000000	add r0, r0

روند اجرای برنامه به این صورت است

14h $\sim$ 18h: r0 = 16
19h: jump to 1Eh
1Eh: r0-= 1
1Fh: if (r0 !=) jump to 19h
.
.
.
20h: r0 += r0
end

به عبارتی، ابتدا رجیستر صفر را مقدار دهی می کند و سپس شروع به کم کردن آن می کند تا صفر شود. نرخ برخورد این برنامه در حافظه نهان دستور رخ می دهد و به این صورت است.

14h -> 010100 -> set 0
18h -> 011000 -> set 1
20h -> 100000 -> set 0

پس ابتدا دستورات ۱۴h $\sim$ ۱۷h لود می شوند، سپس دستورات ۱۸h $\sim$ ۱Fh لود می شوند و در نهایت ۲۰h لود می شود. در این برنامه بخاطر پرش ها هیچ miss ای حاصل نمی شود، زیرا که پرش ها در یک بلاک کش هستند و نیاز به لود کردن بلاک دیگری نیست، پس در این سلسله از دستورات، تنها سه بار miss داریم که باید آن را به تعداد کل دستورات اجرا شده تقسیم کنیم تا miss-rate به دست آید. تعداد کل دستورات نیز برابر است با: ۵ دستور اول که یکبار اجرا می شوند، سپس در یک حلقه می افتیم که ۱۶ بار تکرار می شود و در هر تکرار آن ۳ دستور داریم، و در آخر نیز یک دستور از برنامه باقی می ماند پس:

$$\text{rate miss} = \frac{۳}{۵۴} = \frac{۱}{۱۸} \rightarrow \text{rate hit} = \frac{۱۷}{۱۸}$$

۲. فرض کنید که پردازنده‌ای در اختیار داریم که چهار هسته دارد و از پروتکل MESI برای مدیریت حافظه‌ی نهان بین هسته‌ها استفاده می‌کند. هر هسته یک حافظه‌ی نهان ۲۵۶ بایتی با سیاست نگاشت مستقیم و سیاست write back با بلوک‌های 64 بایتی دارد. همچنین آدرس‌های حافظه‌ی معتبر در سیستم بین 0x10000000 و 0x1FFFFFFF است.

دستورات زیر در هسته‌های مختلف به ترتیب اجرا می‌شوند: (دستور ld برای بار کردن داده حافظه در ثبات فایل و دستور st برای ذخیره‌سازی داده یک ثبات در محلی از حافظه استفاده می‌شوند)

```
ld R1, 0x110000c0 \\ A memory instruction from core 1
st R2, 0x11000080 \\ A memory instruction from core 1
st R3, 0x1FFFFF40 \\ A memory instruction from core 0
ld R4, 0x1FFFFF00 \\ A memory instruction from core 1
st R5, 0x110000c0 \\ A memory instruction from core 1
```

بعد از اجرای دستورات بالا مقادیر کنترلی حافظه‌ی نهان به صورت زیر است:

Final State

Cache 0		
	Tag	MESI state
Set 0	0x1FFFFF	S
Set 1	0x1FFFFF	M
Set 2	0x110000	I
Set 3	0x110000	I

Cache 1		
	Tag	MESI state
Set 0	0x1FFFFF	S
Set 1	0x1FFFFF	I
Set 2	0x110000	M
Set 3	0x110000	E

Cache 2		
	Tag	MESI state
Set 0	0x10FFFF	I
Set 1	0x1FFFFF	I
Set 2	0x10FFFF	M
Set 3	0x10FFFF	M

Cache 3		
	Tag	MESI state
Set 0	0x133333	E
Set 1	0x000000	I
Set 2	0x000000	I
Set 3	0x10FFFF	I

حال با توجه به حالت‌های بالا، جدول زیر که نشان دهنده‌ی حالت اولیه مقادیر کنترلی حافظه‌ی نهان قبل از اجرای دستورات بالا است را تکمیل کنید. در صورتی که در ابتدا آدرس tag نامشخص بود، مقدار X را وارد کنید. برای حالت MESI نیز یکی از حروف M یا E یا S یا I را بنویسید.

Initial State

Cache 0		
	Tag	MESI state
Set 0		
Set 1		
Set 2		
Set 3		

Cache 1		
	Tag	MESI state
Set 0		
Set 1		
Set 2		
Set 3		

Cache 2		
	Tag	MESI state
Set 0		
Set 1		
Set 2		
Set 3		

Cache 3		
	Tag	MESI state
Set 0		
Set 1		
Set 2		
Set 3		

cache 0		
	tag	MESI state
set 0	0x1FFFFF	E
set 1	X	X
set 2	0x110000	(E, I)
set 3	0x110000	(E, I)

cache 1		
	tag	MESI state
set 0	X	X
set 1	0x1FFFFF	(S, I)
set 2	X	X
set 3	X	X

cache 2		
	tag	MESI state
set 0	0x10FFFF	I
set 1	0x1FFFFF	(S, I)
set 2	0x10FFFF	M
set 3	0x10FFFF	M

cache 3		
	tag	MESI state
set 0	0x133333	E
set 1	0x000000	I
set 2	0x000000	I
set 3	0x10FFFF	I

سوال ۳

۳. در درس با مفهوم پیش واکشی[۸] آشنا شدید. برای یافتن آدرس‌هایی که باید پیش واکشی شوند روش‌های گوناگونی وجود دارد که سه مدل پایه‌ای آن به شرح زیر هستند:

- Stride Prefetching
- Stream Prefetching
- Runahead Execution Prefetching

- در مورد این سه روش تحقیق کنید و توضیح دهید هر کدام چگونه تشخیص می‌دهند یک آدرس برای پیش‌واکشی مناسب است یا خیر.
- حال سه برنامه زیر را در نظر بگیرید و توضیح دهید کدام الگوریتم پیش واکشی برای آن‌ها مناسب‌تر از سایرین است:

(آ) اجرای یک الگوریتم گراف که نیازمند چند مرتبه پایش گراف هدف است.

(ب) ضرب داخلی دو ماتریس بزرگ که در حافظه قرار دارند.

(ج) پردازش پرسش و پاسخ‌های متوالی که به یک پایگاه داده ارسال می‌شوند.

- stride prefetching
  در این پیش واکشی، هنگامی که یک miss رخ می دهد، فاصله ی آن تا miss قبلی را بدست می آورد و حدس می زند که پس از این تعداد فاصله دوباره قرار است یک miss با offset مشخص مانند دو miss رخ داده داشته باشیم پس طبق این الگو آدرس بعدی را واکشی می کند.
- stream prefetching
  این نوع پیش واکشی به این معنا است که هرچه به جلو می رویم محتوا های پیش رو را واکشی کنیم، مثلا اگر در حال خواندن خانه های یک آرایه هستیم پس از دیدن دسترسی به خانه های اول و دوم و سوم، باقی دسترسی ها را به صورت پیش واکشی پیش بینی می کنیم.
- Runahead Execution prefetching
  این روش واکشی نیز با نگاه انداختن به درون کل و پیش واکشی بسیار زودتر از رسیدن به آن خط از کد به ما کمک می کند تا miss ها را کاهش دهیم، به این صورت که برنامه را از جاهای بسیار جلوتر از PC اجرا می کند.

- (آ) در اینجا شاهد دسترسی های منظم ولی نه لزوما پشت سر هم هستیم پس stride prefetching ترجیح داده می شود.
- (ب) در این محک، خانه های متوالی حافظه در حال لود شدن هستند پس از روش stream prefetching استفاده می کنیم.
- (ج) در این برنامه، چون access های نامشخص و random به حافظه داریم، پیش خوانی جلو جلو خیلی به ما کمک می کند پس از Runahead Execution prefetching استفاده می کنیم.

سوال ۴

۴. نتیجه‌ی اجرای قطعه کدهای پایتون زیر برای کسانی که با IEEE 754 آشنا نیستند غیرمنتظره خواهد بود:

آ) با استفاده از ماشین حساب‌های دارای قابلیت محاسبه ممیزشناور، نمایش ممیزشناور اعداد 0.1 و 0.2 را در 64 بیت به دست آورید و با جمع آن‌ها نشان دهید که این خطای محاسباتی چه طور به وجود آمده است.

```
print(0.1 + 0.2)                # prints 0.30000000000000004
print(0.1 + 0.2 == 0.3)         # prints False
```

ب) برای اعداد زیر نیز با نوشتن نمایش ۶۴ بیتی عددهای داده شده و انجام جمع نشان دهید که چرا نمی‌توان عدد 9007199254740993 را در استاندارد IEEE 754 نشان داد.

```
print(9007199254740992.0 + 1.0) # prints 9007199254740992.0
```

(آ) برای انجام جمع در اعداد فلوت، ابتدا باید توان آنها را یکی کنیم، سپس مانتیس های آنها را جمع می زنیم.

$$0.1 = 0 \quad 01111111011 \quad 1001100110011001100110011001100110011001100110011010$$
$$0.2 = 0 \quad 01111111100 \quad 1001100110011001100110011001100110011001100110011010$$
$$0.1 + 0.2 = 0 \quad 01111111101 \quad 0011001100110011001100110011001100110011001100110100$$
$$0.3 = 0 \quad 01111111101 \quad 0011001100110011001100110011001100110011001100110011$$

همانطور که مشاهده می کنید، حتی پس از یکسان کردن نماها نیز این دو عدد یکسان نخواهند بود.

(ب) .

$$9007199254740992.0 = 0 \quad 10000110100 \quad 0000000000000000000000000000000000000000000000000000$$
$$1.0 = 0 \quad 01111111111 \quad 0000000000000000000000000000000000000000000000000000$$

فاصله ی توان های این دو عدد دقیقا ۵۲ است به این معنی که عدد کوچکتر در مرحله ی یکسان کردن نماهای آن دو عدد به طور کل از بین می رود. به عبارتی دیگر مانتیس عدد اول به قدری بزرگ نیست که بتواند تمام اطلاعات مورد نیاز این عدد را نمایش دهد.

سوال ۵

۵. فرض کنید که یک نوع نمایش عدد اعشاری وجود دارد که به صورت زیر تعریف می‌شود:

علامت	نما	مانتیس
۱ بیت	۵ بیت	۱۰ بیت

با این توصیف به سوالات زیر جواب دهید:

۱. مقدار بایاس را پیدا کنید.

۲. مقدار $+\infty$ را در این توصیف نمایش دهید.

۳. کوچکترین و بزرگترین عدد نرمال‌شده که می‌توان با این نمایش نشان داد، چه عددی است.

۴. کوچکترین و بزرگترین عدد نرمال‌نشده که می‌توان با این نمایش نشان داد، چه عددی است.

(آ) مقدار بایس برابر است با $۲^{e-۱} - ۱$ اگر e تعداد بیت های نما باشد پس در اینجا بایس برابر است با ۱۵

(ب) .

0000000000 11111 0

(ج) بزرگ ترین عدد مثبت ۰ ۱۱۱۱۰ ۱۱۱۱۱۱۱۱۱۱ است که معادل $۲^{۱۵} * (۲ - ۲^{-۱۰})$
کوچک ترین عدد مثبت ۰ ۰۰۰۰۱ ۰۰۰۰۰۰۰۰۰۰ است که معادل $۲^{-۱۴} * ۲$
بزرگ ترین عدد منفی ۱ ۰۰۰۰۱ ۰۰۰۰۰۰۰۰۰۰ است که معادل $-۲^{-۱۴} * ۲$
کوچک ترین عدد منفی ۱ ۱۱۱۱۰ ۱۱۱۱۱۱۱۱۱۱ است که معادل $-(۲^{۱۵} * (۲ - ۲^{-۱۰}))$

(د) بزرگ ترین عدد مثبت ۰ ۰۰۰۰۰ ۱۱۱۱۱۱۱۱۱۱ است که معادل $۲^{-۱۴} * (۲ - ۲^{-۱۰})$
کوچک ترین عدد مثبت ۰ ۰۰۰۰۰ ۰۰۰۰۰۰۰۰۰۱ است که معادل $۲^{-۲۴}$
بزرگ ترین عدد منفی ۱ ۰۰۰۰۰ ۰۰۰۰۰۰۰۰۰۱ است که معادل $-۲^{-۲۴}$
کوچک ترین عدد منفی ۱ ۰۰۰۰۰ ۱۱۱۱۱۱۱۱۱۱ است که معادل $-(۲^{-۱۴} * (۲ - ۲^{-۱۰}))$

سوال ۶

۶. با توجه به اینکه در استاندارد IEE754 ، یک بیت برای علامت ، ۱۱ بیت برای نما و ۵۲ بیت برای بخش کسری یک عدد ۶۴ بیتی ممیزشناور اختصاص داده شده است،

آ) بزرگ ترین عدد مثبت نرمال و غیر نرمال که قابل نمایش با این فرمت است، چیست؟

ب) کوچکترین عدد مثبت نرمال و غیر نرمال که قابل نمایش با این فرمت است، چیست؟

(آ) نرمال

$$0 \quad 11111111110 \quad 1111111111111111111111111111111111111111111111111111 \rightarrow (2 - 2^{-52}) * 2^{1023}$$

و غیرنرمال

$$0 \quad 00000000000 \quad 1111111111111111111111111111111111111111111111111111 \rightarrow (1 - 2^{-52}) * 2^{-1022}$$

(ب) نرمال

$$0 \quad 00000000001 \quad 0000000000000000000000000000000000000000000000000000 \rightarrow (2) * 2^{-1022}$$

و غیرنرمال

$$0 \quad 00000000000 \quad 0000000000000000000000000000000000000000000000000001 \rightarrow (2^{-52}) * 2^{-1022}$$