تمرین ششم معماری کامپیوتر

دانشکده مهندسی کامپیوتر، دانشگاه صنعتی شریف

آرین احدی نیا

استاد درس: جناب آقای دکتر جهانگیر

دستیار آموزشی: جناب آقای علیپور

پاییز ۱۴۰۰

فهرست عناوین

طراحی واحد کنترل

سوال ۱

دستورات کامپیوتر پایه مانو در صفحه ویکی‌پدیا Mano Machine قابل مشاهده است. این دستورات در حل این مساله برای ما حائز اهمیت است.

Mnemonic	Description
AND	And direct memory to accumulator
ADD	Add direct memory to accumulator (affects carry bit)
LDA	Load direct memory to accumulator
STA	Store accumulator to direct memory
BUN	Unconditionally branch to direct memory
BSA	Store current program counter to direct memory and branch to following address
ISZ	Increment value in direct memory and skip next instruction if the sum is zero
--	Indirect addressing versions of the above instructions
CLA	Clear the accumulator
CLE	Clear the carry bit
CMA	Complement the accumulator
CME	Complement the carry bit
CIR	Circulate accumulator right (through carry bit)
CIL	Circulate accumulator left (through carry bit)
INC	Increment accumulator (does not affect carry bit)
SPA	Skip next instruction if accumulator is positive
SNA	Skip next instruction if accumulator is negative
SZA	Skip next instruction if accumulator is zero
SZE	Skip next instruction if carry bit is zero
HLT	Halt computer by clearing the halt bit latch
INP	Input from character bus to accumulator
OUT	Output from accumulator to character bus
SKI	Skip next instruction if input flag is set
SKO	Skip next instruction if output flag is set
ION	Enable interrupts
IOF	Disable interrupts

(الف) حلقه **for** را می‌توانیم به صورت زیر پیاده‌سازی کنیم. فرض می‌کنیم که مقدار **n** در حافظه‌ای با این برچسب ذخیره شده است. حلقه زیر از یک تا **n** را پیمایش می‌کند.

```
                    // initializing
CLA                 // AC <= 0
STA    i            // MEM[i] <= 0
                    // condition checking
Condition:          // label for condition checking block
LDA    i            // AC <= MEM[i]
CMA                 // AC <= -MEM[i]
ADD    n            // AC <= MEM[n] - MEM[i]
SPA                 // Skip loop break if MEM[n] - MEM[i] > 0
BUN    LoopEnd      // Jump to loop end if MEM[n] - MEM[i] <= 0
LDA    i            // AC <= MEM[i]
INC                 // AC <= MEM[i] + 1
STA    i            // MEM[i] <= MEM[i] + 1
…                   // Loop body, iteration variable is in AC and MEM[i]
BUN    Condition    // Back to condition checking
LoopEnd:            // Label for loop end
```

فرض می‌کنیم که طی چند عملیات متوالی، می‌توانیم مقدار عبارت شرطی را محاسبه و در **accumulator** قرار دهیم. مجموعه این عملیات‌ها را با عبارت

AC <= E.C. (Evaluated Condition)

نمایش می‌دهیم. با استفاده از ساختار شرط در زمان محاسبه مقدار شرط، مقدار آن را برابر صفر یا یک قرار می‌دهیم. مقدار صفر به معنی **False** و یک به معنی **True** خواهد بود.

(ب) حلقه **While** را میتوانیم به شکل زیر پیاده کنیم.

```
Condition:          // label for condition checking block
AC <= E.C.          // Evaluate condition and store it in AC
SPA                 // Skip loop break if AC == 1
BUN   LoopEnd       // Jump to loop end if MEM[n] - MEM[i] <= 0
…                   // Loop body
BUN   Condition     // Back to condition checking
LoopEnd:            // Label for loop end
```

(ج) حلقه **Do Until** را میتوانیم به صورت زیر پیاده‌سازی کنیم.

```
LoopBody:           // Label for loop start
…                   // Loop body
AC <= E.C.          // Evaluate condition and store it in AC
SZA                 // Skip repeat if condition evaluated to zero
BUN   LoopBody      // repeat loop if condition is not zero
```

سوال ۲

بله، به شرط آنکه دستورات RTL سازگار باشند، به صورت الگوریتمی اینکار امکان پذیر است.

توجه بفرمایید که دستورات RTL از چنین نحو زبانی پیروی می‌کنند.

$$Expression: Statement$$

به عنوان مثال:

$$NEQ(R_1, R_2), EQ(R_2, R_3): R_1 \leftarrow R_2 + 1$$

توجه کنید که در حالت کلی، هر یک از رجیسترها در حالات مختلف می‌توانند مقادیر مختلفی بگیرند. یک رجیستر دلخواه R را در نظر بگیرید که دستورات RTL به شکل زیر بر روی آن تعریف شده

$$C_1: R \leftarrow {R_1}'$$

$$C_2: R \leftarrow {R_2}'$$

$$\vdots$$

$$C_k: R \leftarrow {R_k}'$$

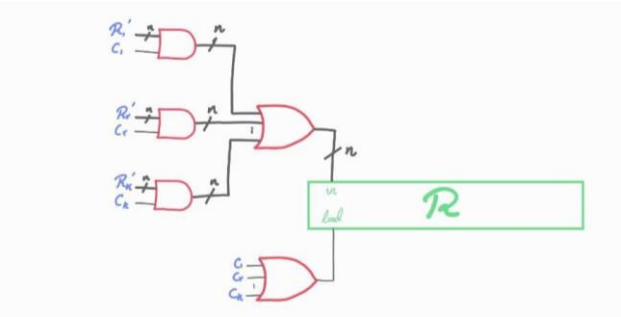
توجه بفرمایید که C_i و ${R_i}'$ زوج مرتبی هستند که در صورت ارضای شرط C_i، مقدار ${R_i}'$ در R قرار می‌گیرد.

با توجه به فرض سازگاری، میدانیم که در صورتی که به ازای یک مجموعه از ورودی، دو شرط C_i و C_j ارضاء شوند، دو مقدار R_i و R_j با یکدیگر برابر هستند. بنابرین میتوانیم با الهام از ایده MUX، به صورت زیر ورودی جدید هر مرحله را محاسبه کنیم و ورودی فعال کننده Load را به صورت زیر در نظر بگیریم.

$$load = \underset{i}{\text{OR}}\, C_i$$

$$R' = \underset{i}{\text{OR}}\, C_i \,.\, {R_i}'$$

توجه کنید که اگر دو شرط همزمان ارضاء شوند، از آنجایی که مقدار پسین آنها برابر است، مقدار نهایی نیز برابر آنها خواهد بود. شماتیک مدار را در شکل زیر می‌توانید ملاحظه فرمایید.



توجه بفرمایید که هر یک از ${R_i}'$ ها برحسب خروجی فعلی ثبات‌ها و به کمک مدارهای ترکیبی قابل محاسبه هستند.

سوال ۳

اجرا زیر برنامه، بخش از روند عادی اجرای برنامه است. به عنوان مثال دستور مانند JMP در MIPS یا CALL در x8086، در حین اجرای برنامه، زیر برنامه را فراخوانی می‌کند و پردازنده آن بخش از دستورات را اجرا می‌کند. در هر بار اجرای برنامه با ورودی‌های مشابه، زیر برنامه‌ها نیز به همان طریق اجرا خواهند شد. اجرای زیر برنامه می‌تواند مقادیر registerها را نیز تغییر دهد. به عبارت دیگر، state را تغییر دهد.

اما اجرای وقفه کاملا متفاوت است. اجرا وقفه توسط سیگنالی خارج از برنامه شروع می‌شود. به عبارت دیگر اجرای وقفه هیچ ارتباطی به اجرای برنامه ندارد. ممکن است در دو دفعه متوالی اجرای برنامه با ورودی‌های مشابه، اجرای وقفه‌ها متفاوت باشد. بر خلاف زیر برنامه که مدیریت اجرای آن به دست برنامه بود، مدیریت اجرا یا عدم اجرای وقفه به دست پردازنده است. منطقا پس از اجرای وقفه مقدار registerها باید دست نخورده باقی بمانند.

سوال ۴

سیگنال‌های کنترلی برابر خواهند بود با

LD_1	Inc_1	ShL_1	ShR_1	LD_2	Inc_2	ShL_2	ShR_2	$Sel\ a$	$Sel\ b$	$Sel\ c$	$Sel\ d$	$pass$	add	and	sub
1	0	0	0	1	0	0	0	1	0	1	0	1	0	0	0
0	0	1	0	0	1	0	0	1	0	1	0	0	0	0	1
1	0	0	0	0	0	0	1	0	1	1	0	0	0	1	0

بنابرین عملیاتی که در هر مرحله اتفاق می‌افتد را می‌توانیم به شکل زیر بیان کنیم.

Reg 1	Reg 2	ALU OP1	ALU OP2	Operation
Load	Load	R2	4D	Pass
Shift Left	Increament	R2	C9	Sub
Load	Shift Right	R1	96	And

در کلاک اول، ALU عملیاتی را انجام نمی‌دهد و عینا ورودی x را خارج میکند. بدیهی است که carry و overflow اتفاق نمی‌افتد. همچنین 4D یک عدد مثبت ناصفر است. بنابرین همه flagها غیرفعال می‌مانند. پس از این مرحله، 4D در هر دو رجیستر R1 و R2 ذخیره میگردند.

در مرحله دوم، تفریق بین R2 و ورودی انجام می‌شود. با توجه به روش مکمل دوم داریم:

$$4D - C9 = 0100\ 1101 - 1100\ 1001 = 0100\ 1101 + 0011\ 0111 = 1000\ 0100$$

توجه بفرمایید که 4D مثبت است و C9 منفی است. بنابرین حاصل تفریق این دو باید مثبت شود. ولی حاصل منفی شده است. بنابرین overflow اتفاق افتاده است. این مورد را حسب آنکه carry وارد شده به بیت علامت از آن خارج نشده نیز میتوان تشخیص داد. همچنین در این جمع carry رخ نمیدهد و حاصل کل منفی است. بنابرین پرچم‌های علامت و سرریز باید فعال شوند. در انتهای این کار، مقدار R1 پس از شیفت برابر 9A و مقدار رجیستر R2 پس از Increament برابر 4E میگردد.

در مرحله بعد مقدار R1 با ورودی And میگردد. حاصل برابر خواهد بود با

$$9A\ \&\ 96 = 1001\ 1010\ \&\ 1001\ 0110 = 1001\ 0010 = 92$$

بدیهی است که سرریز و carry نداریم، حاصل غیر صفر است و منفی، بنابرین تنها پرچم sign فعال می‌گردد. پس از این مرحله مقدار R1 برابر 92 و مقدار R2 برابر 26 میگردد.

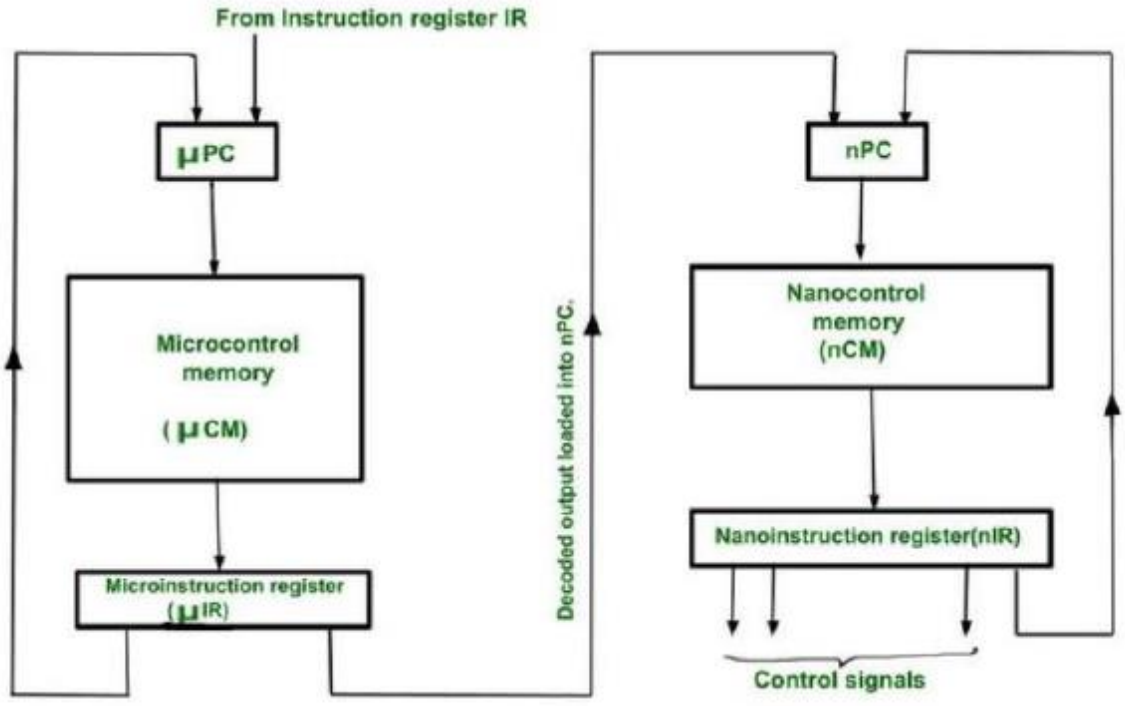
با توجه به توصیفات فوق داریم

R1	R2
4D	4D
9A	4E
92	27

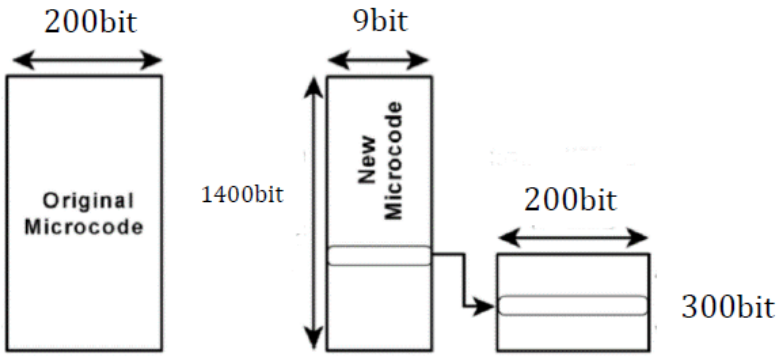
sign	carry	overflow	zero
0	0	0	0
1	0	1	0
1	0	0	0

سوال ۵

می خواهیم با استفاده از حافظه سیگنال های کنترلی را ذخیره و از آن در برنامه استفاده کنیم. دو نوع micro instruction وجود دارد. نوع اول Horizontal است در این روش هر بیت ذخیره شده در حافظه یک سیگنال کنترلی است. نوع دوم vertical است. در این روش هر بیت به طور مستقل سیگنال کنترلی نیست فرض کنید 16 سیگنال کنترلی داریم در روش Horizontal هر خانه حافظه 16 بیتی است اما در روش vertical هر خانه 4 بیتی است و برای بدست آوردن سیگنال های کنترلی یک decoder باید قرار گیرد تا این 4 بیت ا به 16 بیت کنترلی تبدیل کند. وجود decoder باعث کاهش سرعت شده ولی از آن طرف حجم بسیار کمتری حافظه اشغال می کند. Nano programming به معنای ترکیب این دو روش است. این روش را در تصویر زیر مشاهده می کنید.



حال تصویر زیر را مشاهده می کنید.



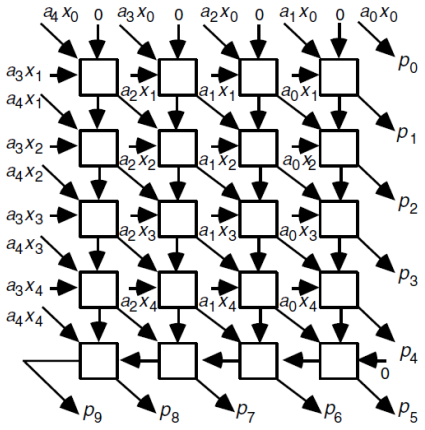
با توجه به اینکه 300 الگو موجود است لذا می توان original microcode بالا را که 1400 خانه ارتفاع آن و 200 بیت پهنای آن است (280000bit) به یک حافظه با ارتفاع 1400 و پهنای 9 بیت که همان vertical است (12600bit) و یک حافظه با ارتفاع 300 و پهنای 200 که همان Horizontal است (60000bit) تبدیل کرد.

$$saving = 280000 - 72600 = 207400bit!$$

حجم اولیه حافظه microprogram (بخش کنترلی) شامل ۲۰۰×۱۴۰۰ بیت است. با توجه به تعداد الگوهای تکراری که ۳۰۰ است، حافظه نانو شامل ۳۰۰ الگوست که در این حالت حافظه ریزبرنامه سازی باید آنها را آدرس‌دهی کند. تعداد بیت‌های این آدرس $\lceil \log_2 300 \rceil = 9$ بیت خواهد شد. پس حجم حافظه ریزبرنامه‌سازی برابر ۹ ×۱۴۰۰ بیت می‌شود. که کاهش ۹۵/۵٪ راه نشان می-دهد. تاخیر مدار در حالت اولیه برابر $1.6\ T_M$ است، با استفاده از حافظه نانو میزان تاخیر به اندازه $1.6\ T_M + T_M = 2.6\ T_M$ خواهد شد. در نتیجه میزان افزایش تاخیر نسبت به حالت اولیه ۶۲,۵٪ است.

عملیات ضرب و تقسیم

سوال ۱

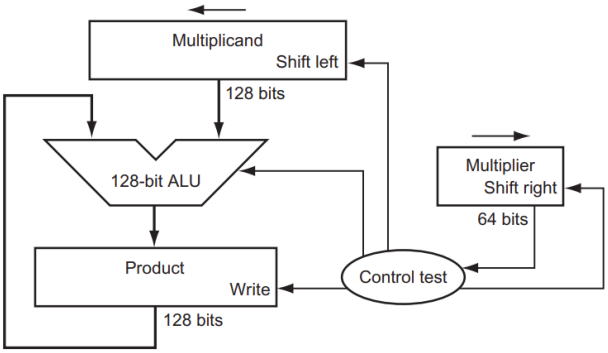


تصویر فوق یک ضرب ۵ در ۵ را نشان می‌دهد. برای محاسبه بیشترین تاخیر باید بلندترین مسیر را در نظر بگیریم. از آنجایی که محاسبه حاصل جمع بیش از رقم نقلی است. سعی می‌کنیم که بیشترین حاصل جمع را در این فرآیند داشته باشیم. بنابرین از بالا سمت چپ شروع می‌کنیم و ردیف یکی مانده به آخر می‌رسیم. از آنجا با محاسبه رقم نقلی به ردیف پایین می‌رویم و به سمت چپ تا محاسبه p_8 حرکت می‌کنیم. در این فرآیند. یک AND، $n-2$ حاصل جمع، $n-1$ رقم نقلی و در انتها دوباره یک حاصل جمع را به توالی محاسبه کردیم بنابرین زمان مورد استفاده برابر است با

$$t_{and} + (n-1)(t_s + t_c) = 10ns + 3(90ns) = 280ns$$

سوال ۲

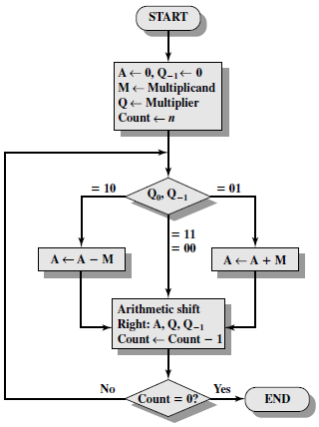
تصویر زیر شمای کلی اجرای این الگوریتم است. که از کتاب مرجع درس اخذ شده.



در جدول زیر، مراحل تعقیب سه رجیستر را ملاحظه می‌فرمایید. همانطور که ملاحظه میفرمایید، در هر مرحله **Multiplier** به سمت راست و **Multiplicand** به سمت چپ شیفت میخورد و اگر بیت کم‌ارزش **Multiplier** برابر ۱ باشد، حاصل با **Multiplicand** جمع می‌شود.

Clock	Multiplicand	Multiplier	Product
0	00000110	1011	00000000
1	00001100	0101	00000110
2	00011000	0010	00010010
3	00110000	0001	00010010
4	01100000	0000	01000010

سوال ۳



تصویر فوق، کلیت کاری که انجام می‌دهیم را معین می‌کند. طبق جدول زیر، مراحل اجرا کار را **trace** می‌کنیم. توجه فرمایید که منظور از **step**، بخش‌های مختلف یک دور اجرای حلقه است. همچنین لطفا توجه بفرمایید که تفریق نیز به کمک جمع با مکمل دوم انجام می‌شود. برای سادگی، **A** را ضرب‌کننده و **B** را ضرب شونده قرار می‌دهیم.

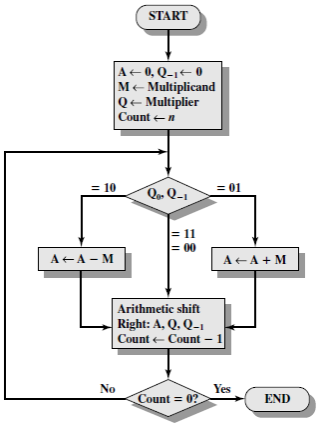
$$M' = 0010\ 0101$$

$count$	$Action$	M	A	Q	Q_{-1}
$initial$	initial	1101 1011	00000000	01010111	0
8	sub	1101 1011	00100101	01010111	0
8	shift	1101 1011	00010010	10101011	1
7	--	1101 1011	00010010	10101011	1
7	shift	1101 1011	00001001	01010101	1
6	--	1101 1011	00001001	01010101	1
6	shift	1101 1011	00000100	10101010	1
5	add	1101 1011	11011111	10101010	1
5	shift	1101 1011	11101111	11010101	0
4	sub	1101 1011	00010100	11010101	0
4	shift	1101 1011	00001010	01101010	1
3	add	1101 1011	11100101	01101010	1
3	shift	1101 1011	11110010	10110101	0
2	sub	1101 1011	00010111	10110101	0
2	shift	1101 1011	00001011	11011010	1
1	add	1101 1011	11100110	11011010	1
1	shift	1101 1011	11110011	01101101	0

بخش پرارزش و کم‌ارزش حاصل هم‌اکنون در **A** و **Q** قرار دارد بنابرین حاصل برابر است با

$$1111\ 0011\ 0110\ 1101 = \sim(0000\ 1100\ 1001\ 0011) = -3219$$

سوال ۴



توجه کنید که در انجام این عملیات، تصمیم ضرب و تقسیم با n زوج مرتب Q_0Q_{-1}، Q_1Q_0، Q_2Q_1 الی $Q_{n-1}Q_{n-2}$ انجام می‌شود و در طی انجام عملیات، این ارقام دست نخورده باقی می‌مانند. بنابرین تعداد عملیات‌های جمع و تفریق مستقل از ضرب‌شونده است.

برای اینکه عمل جمع یا تفریق انجام شود، باید هر یک از این زوج‌ها برابر 10 یا 01 باشد. توجه کنید که در حالت اولیه، Q_{-1} برابر صفر است، بنابرین برای بیشینه کردن تعداد 10ها می‌توانیم از سمت راست یک در میان 1 و 0 قرار دهیم. در این صورت مبنی بر زوج یا فرد بودن n و شروع از صفر یا ۱، چهار حالت بوجود می‌آید.

n	شروع از	Q	تعداد 01	تعداد 10	تعداد کل
زوج	۰	$10 \dots 1010$	$\frac{n}{2}-1$	$\frac{n}{2}$	$n-1$
زوج	۱	$01 \dots 0101$	$\frac{n}{2}$	$\frac{n}{2}$	n
فرد	۰	$010 \dots 1010$	$\frac{n-1}{2}$	$\frac{n-1}{2}$	$n-1$
فرد	۱	$101 \dots 0101$	$\frac{n-1}{2}$	$\frac{n+1}{2}$	n

بنابرین همانگونه که ملاحظه می‌فرمایید، تنها در دو حالت بیشترین تعداد جمع و تفریق بوجود می‌آید.

n	شروع از	Q	تعداد 01	تعداد 10	تعداد کل
زوج	۱	$01 \dots 0101$	$\frac{n}{2}$	$\frac{n}{2}$	n
فرد	۱	$101 \dots 0101$	$\frac{n-1}{2}$	$\frac{n+1}{2}$	n

توجه کنید که تعداد بیش از این امکان پذیر نیست چرا که در مجموع n بار عملیات اجرا می‌گردد. همچنین بدیهی است که حالت دیگر را نمی‌توان برای بیشینه کردن تعداد جمع متصور شد. بنابرین از تمام ترکیب‌های n بیت Q تنها یک حالت تعداد جمع و تفریق را بیشینه می‌کند. بنابرین

$$p = \frac{1}{2^n}$$

می‌باشد.

سوال ۵

(الف) همانگونه که در سوال ۳ ضرب را انجام دادیم، ضرب را انجام می‌دهیم. توجه فرمایید که جهت خلاصه‌نویسی، از نوشتن سطرهای تکراری پرهیز می‌کنیم.

$$M' = 0110\ 1111$$

$count$	$Action$	M	A	Q	Q_{-1}
initial	*initial*	1001 0001	00000000	10010011	0
8	**SUB**	1001 0001	01101111	10010011	0
8		1001 0001	00110111	11001001	1
7	**--**	1001 0001	00110111	11001001	1
7		1001 0001	00011011	11100100	1
6	**ADD**	1001 0001	10101100	11100100	1
6		1001 0001	11010110	01110010	0
5	**--**	1001 0001	11010110	01110010	0
5		1001 0001	11101011	00111001	0
4	**SUB**	1001 0001	01011010	00111001	0
4		1001 0001	00101101	00011100	1
3	**ADD**	1001 0001	10111110	00011100	1
3		1001 0001	11011111	00001110	0
2	**--**	1001 0001	11011111	00001110	0
2		1001 0001	11101111	10000111	0
1	**SUB**	1001 0001	01011110	10000111	0
1		1001 0001	**00101111**	**01000011**	1

حاصل برابر است با

$$0010111101000011 = 2F43 = 12099$$

(ب) توجه بفرمایید که تعداد شیفت برابر تعداد بیت‌های ضرب شونده است. بنابرین در عملیات فوق ۸ بار شیفت انجام می‌گیرد.

در عملیات فوق، دو جمع و سه تفریق انجام شده است. که به تعبیری برابر ۵ جمع و ۳ مکمل‌گیری است. توجه فرمایید که هر سه مکمل حساب شده برابر بودند.

توجه کنید که در انجام این عملیات، تصمیم ضرب و تقسیم با n زوج مرتب Q_0Q_{-1}، Q_1Q_0، Q_2Q_1 الی $Q_{n-1}Q_{n-2}$ انجام می‌شود و در طی انجام عملیات، این ارقام دست نخورده باقی می‌مانند. بنابرین تعداد عملیات‌های جمع و تفریق مستقل از ضرب‌شونده است و تنها با نگاه بر ضرب‌کننده می‌توانیم آن را تشخیص دهیم.

اگر ضرب کننده را قرینه کنیم، برابر میشود با

$$Q' = 0110\ 1101$$

با در نظر گرفتن Q'_{-1}، در کل ۳ زوج 01 و ۳ زوج 10 خواهیم داشت. بنابرین ۶ جمع و ۳ مکمل گیری به همراه ۸ شیفت خواهیم داشت. (10ها: **01101101**0، 01ها 0**11011010**)

سوال ۶

توجه بفرمایید که **overflow** زمانی رخ می‌دهد که خارج‌قسمت به قدری بزرگ باشد که در حافظه مربوطه جا نشود. بیایید مساله را برعکس ببینیم. به رابطه تقسیم زیر توجه کنید.

$$a = bq + r$$

توجه بفرمایید که با توجه به اندازه حافظه **q** و تعریف تقسیم داریم

$$0 \le r < b \Rightarrow 0 \le r \le b - 1$$

$$q < 2^n \Rightarrow q \le 2^n - 1$$

بنابرین خواهیم داشت

$$a \le b(2^n - 1) + b - 1$$

$$\Rightarrow a \le 2^n b - 1$$

$$\Rightarrow a < 2^n b$$

بنابرین در حالتی که **q** و **r** در بیشینه حالت خود باشد، حالت تساوی اتفاق می‌افتد. بنابرین **a** های بزرگتر در این تقسیم منجر به **overflow** می‌شوند. بنابرین

"شرط لازم و کافی برای سرریز خارج‌قسمت n بیتی در تقسیم این است که $a \ge 2^n b$ باشد."

حالت فرمال گزاره‌ای که پیشتر بیان کردیم این است که اگر سرریز در خارج قسمت رخ ندهد، $a < 2^n b$ است. با توجه به عکس نقیض، ثابت کردیم که اگر $a \ge 2^n b$ باشد، سرریز رخ می‌دهد. بنابرین اینکه $a \ge 2^n b$ باشد شرط لازم برای سرریز است. اکنون اثبات می‌کنیم که اگر $a \ge 2^n b$ باشد، سرریز حتما رخ می‌دهد تا کافی بودن این شرط نیز اثبات شود.

برهان خلف، فرض کنید که سرریز رخ نمیدهد. در این صورت $q \le 2^n - 1$ است

$$\begin{cases} a \ge 2^n b \\ a = qb + r \\ 0 \le r < b \\ q \le 2^n - 1 \end{cases} \Rightarrow qb + r \ge 2^n b \Rightarrow (2^n - 1)b + r \ge 2^n b \Rightarrow r \ge b$$

با توجه به تناقض حاصله، حکم اثبات می‌گردد.

ممیز شناور

سوال ۱

(الف) برای نمایش بزرگ‌ترین عدد مثبت، بیشترین مقدار را باید برای نما و بخش کسری در نظر بگیریم. بدیهی است که بیت علامت باید صفر باشد. نما یک عدد ۸ بیتی است. توجه کنید که مقدار FF_{HEX} برای نمایش شبه اعداد رزرو است بنابرین بزرگ‌ترین مقداری که برای نما می‌توانیم در نظر بگیریم برابر FE_{HEX} است که در نمایش **biased** برابر ۱۲۷ میشود. همچنین برای بخش کسری تمام ۲۳ بیت را برابر ۱ در نظر میگیریم. بنابرین عدد مورد نمایش برابر خواهد بود با

$$1.\overbrace{111\ldots111}^{23} \times 2^{127} = (2 - 2^{-23}) \times 2^{127} = 2^{128} - 2^{104} \approx 3.40 \times 10^{38}$$

(ب) برای نمایش کوچک‌ترین عدد مثبت، مقدار نما و کسر را کمترین مقدار ممکنه در نظر میگیریم. توجه کنید که مقدار 00_{HEX} نما برای نمایش شبه اعداد رزرو است بنابرین کوچک‌ترین مقدار نما برابر 01_{HEX} است که در نمایش **biased** برابر ۱۲۶ـ است. همچنین برای بخش کسری نیز تماما صفر خواهد بود. بنابرین عدد مورد نمایش برابر خواهد شد با

$$1.000\ldots000 \times 2^{-126} = 1.17 \times 10^{-38}$$

(ج) در مورد اعداد ناهنجار، از صفحه ویکی‌پدیا Subnormal Number مطالعات انجام شده است.

در نمایش هنجار،

در نمایش هنجار، اعداد به فرم زیر نمایش داده می‌شوند.

$$1.f_1f_2f_3\ldots f_{23}$$

اما در نمایش ناهنجار، اعداد به صورت زیر نمایش داده میشوند.

$$0.f_1f_2f_3\ldots f_{23}$$

در نمایش اعداد ناهنجار به خصوص مقدار نما برابر 00_{HEX} است و معادل **126-** تفسیر می‌گردد. برای اینکه کوچکترین عدد مثبت را نمایش دهیم، تمام بیت‌های بخش کسری به استثنا بیت آخر را صفر در نظر میگیریم. بنابرین عدد مورد نمایش برابر می‌شود با

$$2^{-23} \times 2^{-126} = 2^{-149} = 1.40 \times 10^{-45}$$

(د) کوچک‌ترین تغییر قابل ایجاد در بخش کسری برابر 2^{-23} است. اگر مقدار نما برابر e باشد، فاصله بین دو عدد برابر 2^{e-23} خواهد بود. همانگونه که در بخش ب گفتیم، کوچکترین مقدار نما برابر **126-** است. بنابرین کوچک‌ترین فاصله بین دو عدد هنجار برابر 2^{-149} خواهد بود.

$$2^{-149} = 1.40 \times 10^{-45}$$

(هـ) در بخش الف و ب، بزرگ‌ترین و کوچک‌ترین اعداد مثبت هنجار را محاسبه کردیم. بنابرین بیشترین فاصله میان دو عدد مثبت هنجار برابر خواهد بود با

$$2^{128} - 2^{104} - 2^{-126}$$

(و) توجه بفرمایید که نمای 00_{HEX} و FF_{HEX} برای نمایش اعداد خاص رزرو است. بنابرین برای ۸ بیت نما، ۲۵۴ ترکیب متمایز مجاز وجود دارد. همچنین برای بخش کسری، تمام 2^{23} حالت را میتوانیم در نظر بگیریم. از آنجایی که دو حالت برای علامت وجود دارد، تعداد اعداد با ترکیب این حالات برابر خواهد بود با

$$2 \times (2^8 - 2) \times 2^{23} = 2^{32} - 2^{25}$$

علاوه بر اینها، عدد صفر نیز با بیت‌های تماما صفر قابل نمایش است. بنابرین تعداد اعداد برابر میشود با

$$2^{32} - 2^{25} + 1 = 4261412865$$

همچنین علاوه بر تمام اینها، سه مقدار NaN، $+\infty$ و $-\infty$ نیز قابل نمایش است. بنابرین تعداد کل اعداد قابل نمایش برابر ۴۲۶۱۴۱۲۸۶۸ است.

سوال ۲

بدیهی است که چون عدد منفی است، بیت علامت برابر یک است.

نمایش این عدد در مبنای ۲ برابر است با

$$10111.0\overline{0011}$$

بنابرین در نمایش هنجار، این عدد را میتوانیم به صورت زیر نمایش دهیم.

$$1.01110\overline{0011} \times 2^4$$

نمایش ۴ به صورت **biased** در ۸ بیت به صورت $131 = (83)_{HEX}$ خواهد بود.

همچنین بخش کسری را نیز با $01110\overline{0011}$. پر می‌کنیم.

بنابرین در نهایت عدد به صورت زیر نمایش داده خواهد شد.

1	1	0	0	0	0	0	1	1	0	1	1	1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0

بنابرین این عدد به صورت **Hex** برابر خواهد بود با

$$C1B8\ CCCC$$

سوال ۳

میتوانیم رابطه فوق را به فرم زیر بازنویسی کنیم.

$$\left[\left(\sum_{k=0}^{8}\overline{b_k}2^{8-k}\right)-2^8+1\right]2^{[\sum_{k=9}^{15}b_k2^{k-9}-64]}$$

$$=\left[\left(\sum_{k=0}^{8}(1-b_k)2^{8-k}\right)-2^8+1\right]2^{[\sum_{k=9}^{15}b_k2^{k-9}-64]}$$

$$=\left[\sum_{k=0}^{8}2^{8-k}-\sum_{k=0}^{8}b_k2^{8-k}-2^8+1\right]2^{[\sum_{k=9}^{15}b_k2^{k-9}-64]}$$

$$=\left[2^9-1-\sum_{k=0}^{8}b_k2^{8-k}-2^8+1\right]2^{[\sum_{k=9}^{15}b_k2^{k-9}-64]}$$

$$=\left[2^8-\sum_{k=0}^{8}b_k2^{8-k}\right]2^{[\sum_{k=9}^{15}b_k2^{k-9}-64]}$$

$$=[2^8-bin(b[0:8])]\times 2^{bin(b[15:9])-64}$$

این ساختار یک ساختار ۱۶ بیتی است که ۷ بیت سمت چپ آن نشان دهنده بخش نمایی و ۹ بیت بعدی نشان دهنده بخش کسری عدد است.

نما، ۷ بیت							بخش کسری، ۹ بیت								

توجه بفرمایید که در بخش نمایی، پرارزش‌ترین بیت در سمت چپ است؛ بر خلاف بخش کسری که پرارزش‌ترین بیت در سمت راست است.

نما به صورت نمایش منحرف با انحراف ۶۴ نمایش داده میشود. بنابرین نمای عدد در بازه **64-** الی **63** قرار می‌گیرد. همچنین بخش کسری در بازه در بازه **255-** الی **256** قرار میگیرد.

بنابرین بزرگترین عدد مثبت قابل نمایش برابر است با

$$256\times 2^{63}=2^{71}=2{,}361{,}183{,}241{,}434{,}822{,}606{,}848=2.36\times 10^{21}$$

و کوچکترین عدد منفی برابر خواهد بود با

$$-255\times 2^{63}=-(2^8-1)2^{63}=-2^{71}+2^{63}=-2.35\times 10^{21}$$

سوال ۴

عدد مورد نمایش به صورت زیر خواهد بود.

$$(-1)^{b_{15}} \times 1.\, b_9 b_8 b_7 \dots b_0 \times 2^{bin(b[14:10])-15}$$

بنابرین دو عدد مورد نمایش برابر هستند با

$$A = h7F83 = b0111111110000011 = (-1)^0 \times 1.1110000011 \times 2^{31-15}$$
$$= 1.1110000011 \times 2^{16} = 11110000011000000 = 123072$$

$$B = h6C64 = b0110110001100100 = (-1)^0 \times 1.0001100100 \times 2^{27-15}$$
$$= 1.0001100100 \times 2^{12} = 1000110010000 = 4496$$

$$A + B = 127568 = 11111001001010000 = 1.1111001001010000 \times 2^{16}$$
$$= (-1)^0 \times 1.1111001001 \times 2^{31-15}$$

بنابرین نمایش به صورت

$$0\ 11111\ 1111001001$$

خواهد بود که معادل است با

$$h7FC9$$