تمرین چهارم معماری کامپیوتر

دانشکده مهندسی کامپیوتر، دانشگاه صنعتی شریف

آرین احدی نیا

شماره دانشجویی: [redacted]

استاد درس: جناب آقای دکتر جهانگیر

دستیار آموزشی: جناب آقای علیپور

پاییز ۱۴۰۰

فهرست عناوین

طراحی کامپیوتر پایه

سوال ۱

یکی از انواع دستورات در پردازنده میپس، J-Type است که به صورت زیر است.

6-bit Opcode	26-bit Operand

در این نوع دستورالعمل، مشابه تمام دستورالعمل‌های میپس، ۶ بیت اول مربوط به Opcode است. ۲۶ بیت بعدی، مشخص کننده عملوند این دستورالعمل است.

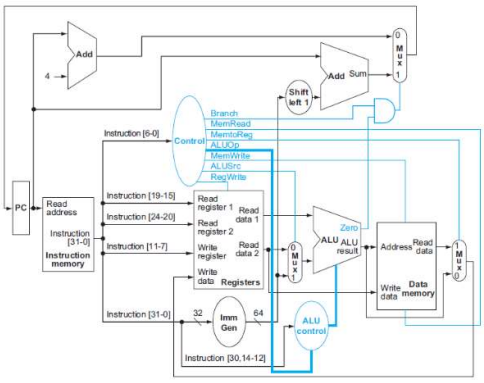
نوعا در پردازنده میپس، تنها دستورات J و JAL از نوع J-Type هستند. بنابرین عملوند مورد نظر آدرس یک خانه در حافظه است که در آن یک دستورالعمل قرار دارد.

توجه کنید که حافظه پردازنده MIPS شامل 2^{32} بایت است. اما به دلیل Alignment، هر ۴ بایت آدرس‌پذیر است و بنابرین دو بیت کم‌ارزش در آدرس‌های این پردازنده برابر صفر است چرا که باید آدرس‌ها بر ۴ بخش‌پذیر باشند. بنابرین با ۳۰ بایت می‌توانیم به طور کامل آدرس یک خانه آدرس پذیر را در پردازنده مشخص کنیم اما، عملوند در این دستور ۲۶ بیتی است. توجه بفرمایید که با توجه به ثابت بودن طول دستورالعمل‌ها، امکان افزایش بیت‌ها وجود ندارد.

دستور JAL و J به ترتیب اکثرا در فراخوانی زیرروال‌ها و پرش‌ها اتفاق می‌افتد. ایده‌ای که طراحان این پردازنده برای حل این مشکل استفاده کردند، این بوده که احتمالا پرش‌ها و فراخوانی تابع‌ها، جایی در نزدیکی همان محل است بنابرین در حافظه نیز دستورالعمل‌های مربوطه نزدیک به دستورالعمل فعلی قرار دارد. بنابرین ۴ بیت پرارزش را برای ساخت آدرس مربوطه، از ۴ بیت پرارزش PC استخراج می‌کنیم.

توجه بفرمایید که این پردازنده دارای دستور JR نیز میباشد که به عنوان عملوند یک رجیستر دریافت میکند و دقیقا به آدرسی که در آن رجیستر قرار گرفته پرش انجام می‌دهد.

سوال ۲



این پردازنده از نوع RISC است و به همین دلیل دستورالعمل‌های آن فرمت به نسبت ساده‌ای دارند. در همه دستورالعمل‌ها، ۶ بیت ابتدایی نمایانگر Opcode است. البته در دستورات نوع R، ۶ بیت انتهایی نیز دستورالعمل را مشخص می‌کنند. بنابرین با توجه به شش بیت ابتدایی، میتوانیم بسیاری از دستورالعمل‌ها را مشخص کنیم و با توجه به ۶ بیت انتهایی نیز وظیفه ALU کاملا مشخص می‌شود.

مزیت این روش این است که میتوانیم با استفاده از یک سخت‌افزار کوچک‌تر که خاص منظوره برای همین کار است، این سیگنال‌ها را مشخص کنیم. این باعث میشود که سرعت عملیات بالا رود. همچنین می‌توانیم از ALU برای کار های دیگر استفاده کنیم. این استفاده دیگر میتواند پردازش مرحله قبل باشد که نوید موازی‌سازی را می‌دهد. اما کاستی این روش آن است

که برای آن نیاز به سخت افزار بیشتر است و به هر طریق هزینه ساخت سخت‌افزار را افزایش می‌دهد. همچنین با افزایش سطح سخت‌افزار، موجب افزایش گرمای تولیدی آن می‌شود.

سوال ۳

از آنجایی که ۱۵ دستور سه آدرسه داریم، برای مشخص کردن Opcode این دستورات دست کم به ۴ بیت نیاز خواهیم داشت. بنابرین در دستورات سه آدرسه، ۴ بیت برای Opcode و ۱۲ بیت برای آدرس‌ها خواهیم داشت.

از ۱۶ ترکیب ممکن برای ۴ بیت Opcode، ۱۵ ترکیب آن را استفاده کرده‌ایم و ۱ ترکیب برای سایر دستورات باقی‌مانده است. چون ۱۴ دستور دو آدرسه داریم، به ۴ بیت دیگر نیاز خواهیم داشته تا Opcode آن را مشخص کنیم. در این صورت ۴ بیت اول را از ترکیب باقی‌مانده از دستور سه آدرسه مشخص می‌کنیم و ۴ بیت بعدی را یکی از ۱۴ ترکیب مورد نظر در نظر میگیریم. به این صورت یک Opcode ۸ بیتی برای این نوع دستور خواهیم داشت. توجه کنید که از ترکیب‌های مختلف این Opcode، ۲ ترکیب دست نخورده باقی مانده است. توجه کنید که ۳۱ دستور ۱ آدرسه داریم. ۸ بیت ابتدایی این نوع دستور، از ترکیب‌های دست نخورده حالت قبلی خواهد بود و تعدادی بیت دیگر برای افزایش تعداد دستورات، نیاز خواهیم داشت. توجه کنید که دو ترکیب برای ۸ بیت اول مجاز خواهد بود بنابرین با اضافه کردن ۴ بیت، میتوانیم ۳۲ ترکیب مختلف داشته باشیم و از ۳۱ عدد از آنها برای این نوع دستور استفاده کنیم و ۱ ترکیب را برای حالت بعد باقی بزایم.

در نهایت ۱۶ دستور بدون آدرس داریم. ۱ ترکیب مجاز برای ۱۲ بیت ابتدایی Opcode داریم. با اضافه کردن ۴ بیت دیگر، ۱۶ دستورالعمل خواهیم داشت.

در نهایت Opcode ها به شکل زیر خواهند بود.

۳ آدرسه	۲ آدرسه	۱ آدرسه	بدون آدرس
0000	11110000	1111111X0000	1111111111110000
0001	11110001	1111111X0001	1111111111110001
0010	11110010	1111111X0010	1111111111110010
0011	11110011	1111111X0011	1111111111110011
0100	11110100	1111111X0100	1111111111110100
0101	11110101	1111111X0101	1111111111110101
0110	11110110	1111111X0110	1111111111110110
0111	11110111	1111111X0111	1111111111110111
1000	11111000	1111111X1000	1111111111111000
1001	11111001	1111111X1001	1111111111111001
1010	11111010	1111111X1010	1111111111111010
1011	11111011	1111111X1011	1111111111111011
1100	11111100	1111111X1100	1111111111111100
1101	11111101	1111111X1101	1111111111111101
1110		1111111X1110	1111111111111110
		111111101111	1111111111111111

اگر تعداد بیت‌های لازم برای آدرس‌دهی همه آدرس‌ها را برابر بگیریم، برای هر آدرس نیز ۴ بیت خواهیم داشت و شکل دستورات به شکل زیر خواهد شد.

OPCODE 4bit	OP1 4bit	OP2 4bit	OP2 4bit

OPCODE 8bit	OP1 4bit	OP2 4bit

OPCODE 12bit	OP1 4bit

OPCODE 16bit

سوال ۴

اگر در نظر بگیریم که مقادیر Load شده در یک رجیستر مانند AC قرار می‌گیرند، وضعیت به شرح زیر خواهد بود.

با اجرای دستور

```
LdI 20
```

مقدار ۲۰ که در دستورالعمل قرار گرفته است در AC قرار میگیرد. (AC = 20)

با اجرای دستور

```
LdA 20
```

محتوای آدرس ۲۰ حافظه که برابر ۴۰ است در AC قرار میگیرد. (AC = 40)

دستور Load Indirect به این صورت است که در حافظه‌ای که در دستور مشخص شده، آدرس حافظه‌ای که مقدار مورد نظر در آن قرار دارد، قرار دارد. بنابرین با اجرای دستور

```
LdInd 20
```

به مقداری که در آدرس ۲۰ قرار گرفته رجوع می‌کنیم. این مقدار برابر ۴۰ است. بنابرین مقداری که در حافظه ۴۰ قرار دارد را در AC قرار میدهیم که برابر ۶۰ است. (AC = 60)

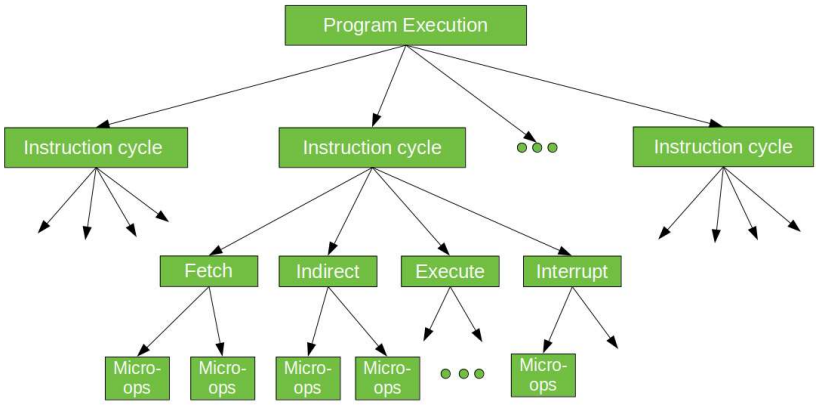
اجرای دستور

```
LdInd 30
```

نیز مشابه است. مقدار در حافظه ۳۰ برابر ۵۰ و مقدار در حافظه ۵۰ برابر ۷۰ است. بنابرین مقدار ۷۰ در حافظه قرار می‌گیرد. (AC = 70)

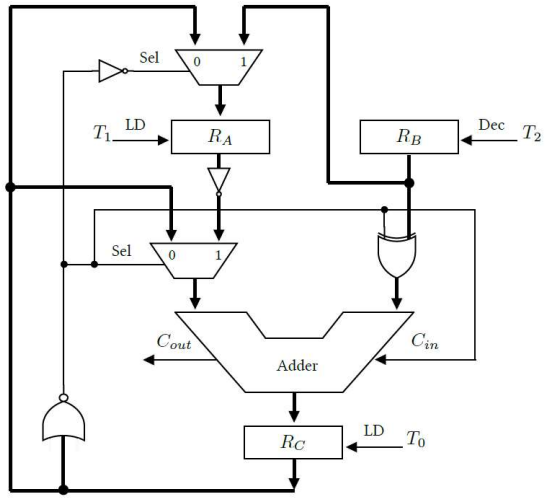
سوال ۵

آن طور که بنده متوجه شده‌ام، منظور از μop همان ریز دستوراتی است که در هر سیکل اجرا می‌شود. مشابه سوال ۷، این دستورات را می‌توانیم با کد RTL پیاده‌سازی کنیم.



تصویر ۱، مرجع: GeeksForGeeks

بنابرین نظیر به ورودی‌ها، کد RTL مورد نظر را تولید می‌کنیم.



$$
\begin{aligned}
T_0, NOR(R_C) &: R_C \leftarrow -R_A - R_B - 1 \\
T_0, OR(R_C) &: R_C \leftarrow R_C + R_B \\
T_1, NOR(R_C) &: R_A \leftarrow R_C \\
T_1, OR(R_C) &: R_A \leftarrow R_B \\
T_2 &: R_B \leftarrow R_B - 1
\end{aligned}
$$

حتما شبه کد این سوال را در دیگر پاسخ ها نگاه کنید

توجه کنید که دستور اول با توجه به روابط مکمل دوم ساده‌سازی شده‌است.

$$\sim R_A + \sim R_B + 1 = (\sim R_A + 1) + (\sim R_B + 1) - 1 = -R_A - R_B - 1$$

سوال ۶

(آ) مستقل از دستورالعمل‌ها سخت افزار را جز به جز تشریح میکنیم.

توجه بفرمایید که ورودی‌های S_i که مربوط به ورودی‌های کنترلی MUXها هستند و ورودی‌های LD که برای فعال‌سازی بارگیری ثبات‌ها هستند، توسط Control Unit با توجه به فاز اجرا و دستور مورد نظر مقداردهی می‌شوند.

ALU این پردازنده مانند بسیاری از پردازنده‌های دیگر دو ورودی و یک سری خطوط کنترلی که با حرف $F_{2:0}$ مشخص شده است دارد. یکی از ورودی‌های این ALU همواره MBR و ورودی دیگر حسب دستورالعمل، MBR و یا AC است.

توجه کنید که به طور غیر مستقیم، قابلیت صفر کردن مقدار AC را داریم. به این صورت که S_5 را برابر 1 قرار می‌دهیم تا هر دو ورودی ALU مقدار MBR باشد. سپس با استفاده از $F = 001$ که مربوط به دستور تفریق است، این مقدار را از خودش کم می‌کنیم تا مقدار صفر به دست آید و با فعال کردن ورودی LD ثبات AC، صفر را در AC قرار می‌دهیم.

همچنین این پردازنده دارای سه رجیستر کمکی نیز میباشد که داده‌های آنها نیز قابل انتقال به MBR است. همچنین این داده‌ها به صورت غیر مستقیم و به واسطه MBR به AC نیز میتوانند منتقل شوند. به این صورت که ابتدا آنها را به MBR منتقل کنیم و مقدار صفر را به نحوی که پیشتر گفته شد در AC قرار می‌دهیم و سپس با قرار دادن $F = 000$، $S_5 = 0$ و $LD_{AC} = 1$ حاصل صفر به علاوه مقدار مورد نظر را محاسبه و در AC ذخیره می‌کنیم. به این صورت مقدار هر یک از سه ثبات را می‌توانیم در AC یا MBR ذخیره کنیم.

مرحله Write back به دو صورت انجام میگردد.

۱. اگر بخواهیم نتیجه را در حافظه بنویسم، باید آنرا در MBR قرار دهیم و آدرس حافظه را در MAR قرار دهیم.
۲. اگر بخواهیم نتیجه را در ثبات بنویسیم، باید آن را در AC قرار دهیم و بارگیری ثبات مورد نظر را فعال کنیم.

انتقال داده از AC به MBR از طریق هر از ثبات‌ها و به طور ویژه از طریق R_T می‌تواند انجام شود. به این صورت که فعال سازی ورودی LD این ثبات، داده AC در آن قرار بگیرد و سپس با فعال سازی ورودی LD ثبات MBR و قرار دادن $S_4S_3S_2 = 100$، این مقدار را در MBR قرار دهیم. در واقع به نحوی نوشتن در حافظه از مراحل نوشتن در ثبات نیز عبور می‌کند.

در فاز اول اجرای دستورالعمل، باید آن را واکشی کنیم. واحد کنترل S_1S_0 را برابر 00 قرار می‌دهد تا مقدار PC در MAR قرار بگیرد. در مرحله بعدی دستور از حافظه بارگیری میشود و ورودی INC برای PC فعال میشود تا به دستور بعدی برسیم. در انتهای این مرحله دستور در MBR قرار می‌گیرد و مقدار PC نیز بروزرسانی می‌گردد. سپس در مرحله بعد ورودی LD ثبات IR فعال می‌گردد تا دستورالعمل از MBR در آن قرار بگیرد.

هر یک از عملوندهای دستورالعمل ها ممکن است از ثبات‌ها و یا حافظه آدرس دهی شوند. با توجه به مواردکه پیشتر گفتیم، به شکل زیر با هر یک از آنها می‌توانیم برخورد کنیم.

۱. اگر هر دو از حافظه بودند: ابتدا یکی از در MBR و سپس AC قرار می‌دهیم. سپس دیگر را در MBR قرار می‌دهیم.
۲. اگر هر دو از ثبات‌ها بودند: ابتدا یکی را به نحوی که پیشتر گفتیم در AC قرار می‌دهیم و سپس دیگر را در MBR قرار می‌دهیم.
۳. اگر یکی از ثبات‌ها و دیگر از حافظه باشد: حسب ترتیب احتمالی (که برای دستورالعمل‌هایی مانند sub اهمیت دارد) یکی را در AC و دیگر را در MBR قرار می‌دهیم.

اگر دستورالعمل مورد نظر تک عملوندی باشد، آن را در MBR قرار می‌دهیم چرا که اولا هر دو ورودی ALU به MBR دسترسی دارند، ثانیا مسیر قرارگیری در MBR در هر صورت کوتاه‌تر از قرار گیری در AC است.

(ب) مراحل اجرای دستورالعمل مورد نظر در ادامه نوشته شده است. در هر خط، دستورالعمل‌هایی که در یک پالس اجرا میشوند را نوشته‌ایم. به پاسخ اقای غضنفری مراجعه شود

$MA \leftarrow PC$

$MBR \leftarrow MEM[MAR]$

$IR \leftarrow MBR$

$MAR \leftarrow OPERAND$

$AC \leftarrow ALU_{SUB}(MBR, MBR)$ `(AC = 0)`

$MBR \leftarrow MEM[MAR]$

$AC \leftarrow ALU_{ADD}(MBR, AC)$ `(AC = MBR)`

$MBR \leftarrow R_0$

$AC \leftarrow ALU_{ADD}(MBR, AC)$

$R_0 \leftarrow AC$

بنابرین همانگونه که ملاحظه می‌فرمایید به ۷ پالس ساعت برای اجرای این دستورالعمل احتیاج داریم. توجه بفرمایید که به دلیل امکان بوجود آمدن Race، امکان موازی سازی بین خواندن و نوشتن از یک ثبات وجود ندارد.

سوال ۷

(آ) همانگونه که ملاحظه می‌فرمایید، دستورالعمل‌های این ماشین همگی یک فرمت دارند؛ به این صورت که ۴ بیت ابتدایی مربوط به Opcode و ۱۲ بیت باقی‌مانده مربوط به آدرس است. دو حالت برای مشخص کردن تعداد دستورالعمل‌های این ماشین در نظر می‌گیریم:

۱. اگر چنین مد نظر باشد که Opcode تنها در همان بخش ۴ بیتی مشخص شود، حداکثر $2^4 = 16$ دستورالعمل مختلف خواهیم داشت.
۲. حالت تعمیم یافته چنین است که برای دستورالعمل‌هایی که Operand ندارند، بخش ۱۲ بیتی آدرس را نیز برای Opcode در نظر بگیریم. در این صورت اگر m دستور آدرس‌دار داشته باشیم، $(2^4 - m)(2^{12})$ دستور بدون آدرس میتوانیم داشته باشیم که در مجموع برابر $2^{16} - (2^{12} - 1)m$ دستور خواهد بود.

برای این ماشین آدرس‌دهی‌های مختلفی میتوان در نظر گرفت. در ادامه به برخی از آنها اشاره می‌کنیم.

۱. آدرس‌دهی مستقیم: با قرار دادن آدرس مورد نظر در MAR، داده مورد نظر را میتوانیم در MBR دریافت کنیم. سپس میتوانیم آن را در AC قرار دهیم.
۲. آدرس‌دهی غیرمستقیم: با قرار دادن آدرس غیرمستقیم در MAR، آدرس اصلی در MBR قرار خواهد گرفت. سپس MBR را در MAR قرار می‌دهیم و دوباره از حافظه مقدار جدید در MBR دریافت می‌کنیم. این مقدار برابر همان مقدار مورد نظر است. سپس این مقدار را در AC قرار میدهیم.

۳. آدرس‌دهی آنی: میتوانیم به جای آدرس ۱۲ بیتی در دستورالعمل، یک مقدار ۱۲ بیتی در آن قرار دهیم. البته توجه داشته باشید که محاسبات این کامپیوتر ۱۶ بیتی است و نمیتوانیم انطباق کامل را داشته باشیم اما برای بعضی مصارف مانند جمع با اعداد کوچک یا Increament، میتواند مفید باشد.

بدیهی است که برخی از انواع آدرس‌دهی مانند Base Addressing و آدرس دهی ثباتی به دلیل عدم وجود سخت افزار مربوط امکان‌پذیر نیست.

(ب) توجه بفرمایید که طی فرآیند Fetch، دو کار باید انجام شود.

$$IR \leftarrow MEM[PC]$$

$$PC \leftarrow PC + 1$$

بنابرین این فرآیند را می‌توانیم به این صورت در نظر بگیریم. توجه کنید که T_i منظور گام زمانی است و توالی را ضمن اجرای دستورات RTL مشخص می‌کند.

$T_0: MAR \leftarrow PC$

$T_1: MBR \leftarrow MEM[MBR]$

$T_1: PC \leftarrow PC + 1$

$T_2: IR \leftarrow MBR$

برای مرحله Decode، باید آدرس مورد نظر در صورت وجود در MAR قرار بگیرد.

$T_3: MAR \leftarrow IR[11:0]$

(ج) دستورات مورد نظر به زبان RTL به صورت زیر خواهد بود. توجه بفرمایید که برای دستورات زیر از مرحله Fetch و Decode که پیشتر به آن اشاره کردیم صرف نظر شده و مراحل بعدی آورده شده است. توجه بفرمایید برای دستوراتی که عملوند ندارند، نیازی به اجرای مرحله فوق نیست. برای دستور Halt، PC را به برنامه بالاسری، که اجرای این برنامه را شروع کرده بر میگردانیم. از آنجایی که برنامه اصلی از خانه صفر شروع میشود و آدرس بازگشت نیز در اولین خانه ذخیره می‌گردد، باید خانه صفرم را در PC قرار دهیم.

Load	$T_4: MBR \leftarrow MEM[MAR]$
	$T_5: AC \leftarrow MBR$

Store	$T_4: MBR \leftarrow AC$ $T_5: MEM[MAR] \leftarrow MBR$
Add	$T_4: MBR \leftarrow MEM[MAR]$ $T_5: AC \leftarrow ALU_{ADD}(AC, MBR)$
Subt	$T_4: MBR \leftarrow MEM[MAR]$ $T_5: AC \leftarrow ALU_{SUB}(AC, MBR)$
Input	$T_3: AC \leftarrow InReg$
Output	$T_3: OutReg \leftarrow AC$
Halt	$T_3: MAR \leftarrow 0$ $T_4: MBR \leftarrow MEM[MAR]$ $T_5: PC \leftarrow MBR$

سوال ۸

برای حل این مساله از الگوریتم رای اکثریت Boyer-Moore که در درس ساختمان داده‌ها و الگوریتم‌ها با آن آشنا شدیم، استفاده می‌کنیم. شبه کد این الگوریتم به این صورت است.

```
candidate = 0
count = 0
for i <- 1 to n
        if count == 0
                candidate = V[i]
                count += 1
        else if V[i] == candidate
                count += 1
        else
                count -= 1
count = 0
for i <- 1 to n
        if V[i] == candidate
                count += 1
```

این الگوریتم دوبار روی دیتا پیمایش انجام می‌دهد. بار اول کاندید احتمالی Majority را معرفی می‌کند و بار دوم تعداد دقیق آرای آن کاندید را میشمارد.

به منظور استفاده بهتر از حافظه، نمره‌ها را ذخیره نمی‌کنیم و برای هر بار پیمایش یک بار نمره را از اساتید میپرسیم. بنابرین دقیقا دوبار نمره از اساتید جمع آوری میشود. توجه بفرمایید که حفظ ترتیب برای این دوبار پیمایش اهمیتی ندارد.

مقدار candidate یکی از حالات ممکن نمره فرد است و مقدار count تعداد تکرار آن را نمایش می‌دهد. بنابرین candidate باید تعداد بیت کافی برای نمایش نمره را داشته باشد. اگر فرض کنیم که نمره یک مقدار صحیح بین صفر تا ۱۰۰ دارد، برای نمایش آن ۷ بیت کافی است و اگر تعداد اساتید نمره دهنده n باشد، تعداد تکرار یک نمره را میتوان در $\lceil \log n \rceil$ بیت ذخیره کرد.

پس از باز اول که نمره واصل شد، آن نمره را به عنوان $V[i]$ در نظر میگیریم و بلوک حلقه اول را به ازای آن $V[i]$ اجرا میکنیم. برای اینکه برای ذخیره $V[i]$ آن لحظه حافظه داشته باشیم، یک متغیر ۷ بیتی دیگر نیز ممکن است که نیاز داشته باشیم تا ذخیره کنیم.

برای اجرای حلقه دوم نیز دوباره نمره را از اساتید دریافت میکنیم و در لحظه واصل شدن، بلوک را به ازای آن نمره انجام می‌دهیم.

در نهایت اگر اکثریتی وجود داشته باشد، مقدار آن در candidate و تعداد تکرار آن در count قرار میگیرد. اگر اکثریتی نباشد، هر دوی این متغیر ها مقدار خواهند داشت اما طبیعتا مقدار count کمتر از نصف تعداد خواهد بود. در نهایت برای اینکه پاسخ قطعی به پرسش داده شده بدهیم، باید تعداد آرا را در حافظه داشته باشیم تا در نهایت بتوانیم مقایسه انجام دهیم.

اگر تعداد اساتید برابر m باشد برای ذخیره count به $\lceil \log m \rceil$ بیت احتیاج داریم. بنابرین در مجموع به $\lceil \log m \rceil + \lceil \log n \rceil$ بیت احتیاج داریم. در صورتی که برای پردازش رای که در آن لحظه در حال بررسی هستیم نیاز به حافظه مجزا داشته باشیم، $\lceil \log n \rceil$ دیگر باید به اندازه حافظه اضافه کنیم اگر نیاز باشد که بگوییم بیش از نصف اساتید رای یکسان داشته‌اند، باید حداقل نصف تعداد اساتید را در حافظه ذخیره کنیم که نیاز به $\lceil \log m \rceil - 1$ بیت دیگر دارد.

توجه کنید که اگر تعداد آرای یک کاندید بیش از نصف شود، در همان لحظه می‌توانیم آن را به عنوان خروجی اعلام کنیم، بنابرین برای ذخیره count میتوانیم از $\lceil \log m \rceil - 1$ بیت استفاده کنیم. بنابرین در مجموع نیاز به $\lceil \log m \rceil + \lceil \log n \rceil - 1$ بیت خواهیم داشت.