دوره هفدهم، شماره بهار و تابستان ۱۴۰۳

مجله فناوری اطلاعات در طراحی مهندسی

Information Technology in Engineering Design

https://sanad.iau.ir/journal/ited

شناسایی و تشخیص خطا در محاسبات دی‌ان‌ای مبتنی بر مدل ادلمن-لیپتون

فرزانه فاموری[۱] امیر صباغ ملا حسینی*[۲] آزاده سادات عمرانی زرندی[۳]

(۱) گروه مهندسی کامپیوتر، واحد کرمان، دانشگاه آزاد اسلامی، کرمان ایران

(۲) گروه مهندسی کامپیوتر، واحد کرمان، دانشگاه آزاد اسلامی، کرمان ایران*

(۳) گروه مهندسی کامپیوتر، دانشگاه شهید باهنر کرمان، کرمان، ایران

(تاریخ دریافت :۱۴۰۲/۱۱/۱۶ تاریخ پذیرش:۱۴۰۳/۰۳/۲۶)

چکیده

محاسبات DNA حوزه‌ای از محاسبات طبیعی است و بر اساس این ایده است که فرآیندهای زیست‌شناسی مولکولی، می‌تواند برای عملیات حسابی و منطقی روی اطلاعات رمزگذاری‌شده به عنوان رشته‌های DNA استفاده شود. DNA معمولاً در لوله‌های آزمایشی که مستعد خطا هستند، محاسبه می‌شود. سیستم عددی منطبق‌شده، برای سادگی و قابلیت اطمینان فرایندهای محاسباتی DNA حائز اهمیت است. سیستم اعداد مانده‌ای، انتخاب خوبی برای قابل اعتماد کردن و کارآمدتر کردن عملیات محاسباتی DNA است. قابلیت‌های سیستم تشخیص و تصحیح خطای RNS را می‌توان برای محاسبات DNA مستعد خطا به کار برد. در این مقاله، یک سیستم محاسباتی DNA را بر اساس سیستم اعداد مانده‌ای افزونه (RRNS) پیشنهاد شده است. این سیستم می تواند دو خطا را تشخیص و یک خطا را نیز تصحیح کند. مدل ادلمن-لیپتون برای انجام عملیات DNA با قابلیت تشخیص و تصحیح خطا استفاده شده است. مزیت این سیستم محاسباتی پیشنهادی، توانایی تشخیص و تصحیح خطاها است. با این حال، عملیات محاسباتی پیشنهادی روی اعداد بزرگتر ارائه شده توسط DNA کار می کند، که RNS این اعداد را به اعداد کوچکتر تقسیم می کند. اجرای عملیات حسابی بر روی این اعداد کوچک، احتمال خطا در عملیات DNA را کاهش می دهد.

کلمات کلیدی: محاسبات DNA، سیستم اعداد مانده‌ای افزونه (RRNS)، تشخیص خطا، تصحیح خطا

*عهده‌دار مکاتبات:

امیر صباغ ملا حسینی

نشانی: گروه مهندسی کامپیوتر، واحد کرمان، دانشگاه آزاد اسلامی، کرمان ایران

پست الکترونیکی:**sabbagh@iauk.ac.ir**

۱- مقدمه

محاسبات مولکولی یا محاسبات DNA (Deoxyribose Nucleic Acid) یکی از رشته‌های جدید علوم کامپیوتر است. این یک روش جدید برای محاسبات موازی انبوه است که می‌تواند در کسری از زمان به حل مسائل-NP کامل یا غیرقطعی چند جمله‌ای زمان بپردازد. مسائل ترکیبی، حوزه دیگری است که محاسبات DNA در آن برتری دارد. این ترکیبی عالی از بیوشیمی، زیست‌شناسی مولکولی و علوم کامپیوتر است که به محققان اجازه می‌دهد، عملیات‌های حسابی و منطقی را انجام دهند که محاسبات را با استفاده از مولکول‌های بیولوژیکی به جای پردازنده‌های سیلیکونی معمولی انجام می دهد[۱]. محاسبات DNA یک زمینه نوظهور است که می‌تواند مسئول گسترش سایر فناوری‌های امیدوارکننده باشد. نقش حیاتی محاسبات DNA به دلیل ذخیره سازی بالا، کارایی انرژی و قابلیت‌های موازی در مقیاس بزرگ است. چنین قابلیت‌های امیدوارکننده‌ای تمرکز بسیاری از محققان را برای جایگزینی رایانه‌های سنتی، یعنی رایانه‌های مبتنی بر سیلیکون با رایانه‌های بیومولکولی، یعنی رایانه‌های مبتنی بر DNA، به خود جلب کرده است. این کار چالش‌های بلادرنگ مختلفی را که محققان مختلف برای پیاده‌سازی محاسبات DNA در زمینه‌های کاربردی مختلف با آن مواجه هستند، مورد بحث قرار می‌دهد. این شامل چالش‌های مبتنی بر شیمی DNA، احتمال خطا و هزینه، مسائل ترکیبی و حالت محدود است. زمینه‌های کاری محاسبات DNA در آینده، قابلیت‌های آن را برای حل چالش‌های مختلف فناوری‌های عصر جدید مانند داده‌های بزرگ، محاسبات ابری، بلاک چین، محاسبات کوانتومی، نانوتکنولوژی و بسیاری دیگر نشان می‌دهد[۲]. محاسبات DNA یک فناوری نوظهور است که می تواند بر محدودیت‌های فناوری محاسبات مبتنی بر سیلیکون به دلیل سرعت محاسبات بالا و قابلیت‌های موازی‌سازی غلبه کند[۳].

در سال ۱۹۶۱، فیزیکدان ریچارد فاینمن برای اولین بار ایده استفاده از سلول‌های زنده و مجموعه پیچیده مولکولی در ساخت کامپیوترهای میکروسکوپی را مطرح کرد. فاینمن درباره مشکل کار کردن و کنترل کردن اجزا در ابعاد و مقیاس کوچک بحث کرد و رشته نانوتکنولوژی را بنیان نهاد. اگر چه او به صورت عمده روی ذخیره اطلاعات و کار در سطح مولکولی تمرکز کرد، به پتانسیل سیستم‌های بیولوژیکی به عنوان پردازنده‌های اطلاعات در مقیاس کوچک اشاره کرد [۴]. در سال ۱۹۹۴، ادلمن با رمزگذاری رئوس و لبه‌ها با استفاده از رشته‌های DNA، انجام یک سری عملیات مولکولی بر روی محلول DNA و در نهایت رمزگذاری خروجی، مسئله هامیلتونی را در یک لوله آزمایش حل کرد [۵].

بسیاری از تحقیقات پیرامون DNA امروزه به رفع مسائل تحقیقات ترکیبی جست‌وجو معطوف شده است. به‌هرحال برای اینکه محاسبات DNA در محدوده گسترده‌تری از مشکلات کاربرد داشته باشد، به عملیات ریاضی پایه از قبیل عملیات منطقی مثل AND، OR و NOT، و عملیات حسابی مثل جمع و تفریق، ضرب نیاز است که بتوان در طراحی

ALU مبتنی بر DNA برای ساخت کامپیوترهای مولکولی از آن استفاده نمود. ریاضیات نقش مهمی در عملکرد و کارایی یک کامپیوتر بازی می کند. ساختن پلتفرم‌های محاسباتی جدید با پشتیبانی از محاسبات باینری سنتی و فن‌آوری‌های مبتنی بر سیلیکون برای برآورده کردن الزامات برنامه‌های امروزی، بدون توجه به اینکه دستگاه‌های تعبیه‌شده یا رایانه‌های با کارایی بالا را در نظر می‌گیریم، به طور فزاینده‌ای چالش برانگیز می‌شود. اگرچه محاسبات باینری با موفقیت برای طراحی سیستم‌های محاسباتی مبتنی بر سیلیکون استفاده شده است، ماهیت موقعیتی این نمایش، منجر به انتشار رقم نقلی می‌شود؛ که از قابلیت موازی‌سازی در ابتدایی‌ترین سطوح جلوگیری می‌کند و منجر به مصرف انرژی بالا می‌شود. از این رو، تحقیق بر روی یک سیستم عددی برای کشف موازی‌سازی و بهره‌گیری از ویژگی‌های فناوری‌های نوظهور برای بهبود عملکرد و بازده انرژی سیستم‌های محاسباتی بسیار مورد توجه است [۶].

در محاسبات DNA ، به دلیل تأثیر انتشار رقم نقلی، داشتن یک سیستم عددی که با اصول سادگی و موازی‌سازی در محاسبات DNA همسو باشد، مهم است. استفاده سیستم اعداد مانده ای (RNS[1]) در عملیات ریاضی DNA باعث کارآمدی DNA می‌شود. RNS به طور بالقوه می‌تواند الزامات قابلیت اطمینان را با استفاده از ویژگی موازی‌سازی و عملیات حسابی پیمانه‌ای کاهش دهد. در این سیستم، محاسبات روی پیمانه‌ها به صورت جداگانه انجام می‌شود. اگر یکی از پیمانه‌ها خطا داشته باشد، اثر آن به پیمانه دیگر منتقل نمی‌شود و همچنین مشکل انتشار رقم نقلی را ندارد [۷]. علاوه بر این، DNA معمولاً در لوله‌های آزمایشی که مستعد خطا هستند محاسبه می‌شود و به دلایل ناشناخته مانند واکنش‌های بیوشیمیایی مثبت و منفی کاذب، ممکن است نتایج نادرستی رخ دهد [۹و۸]. اگرچه انگیزه اولیه در مطالعات RNS اجرای عملیات محاسباتی سریع است، اما برخی از ویژگی‌های جالب مرتبط با این سیستم وجود دارد که آن را برای تشخیص و تصحیح خطا ایده‌آل می‌کند [۷]. قابلیت‌های تشخیص و تصحیح خطای RNS را می‌توان برای محاسبات DNA مستعد خطا به کار برد. برای اولین بار در[۱۰] روش‌های پیشنهادی برای انجام عملیات حسابی در RNS با استفاده از رشته‌های DNA ارائه شد. در [۹] عملیات تشخیص فقط یک خطا مبتنی بر RRNS[2] برای محاسبات DNA انجام شده است. در در مقاله [۱۱] عملیات تشخیص و تخصیص خطا مبتنی بر RRNS برای محاسبات DNA برای تشخیص دو خطا و تصحیح یک خطا بر اساس مدل استیکر[3] انجام داده‌ایم.

در اینجا یک سیستم محاسباتی DNAای پیاده‌سازی کرده‌ایم که قابلیت موازی‌سازی با سرعت بالا و قابلیت تحمل‌پذیری خطا مبتنی بر سیستم اعداد مانده‌ای را دارد. مشخصات و ویژگی‌های این سیستم به این صورت است. جهت

[1] Residue Number System
[2] Redundant RNS
[3] Sticker

انجام دستورات DNA از مدل آدلمن – لیپتون[1] استفاده شده است. عملیاتی ریاضی که عملیات تشخیص و تصحیح خطا روی آنها انجام شده است، عمل جمع است. برای اینکه این سیستم بتواند دو خطا را تشخیص و یک خطا را تصحیح کند، از سیستم اعداد مانده‌ای افزونه با مجموعه پیمانه ۴تایی و محدوده دینامیکی 6n بیتی استفاده شده است که دوتای این پیمانه‌ها، پیمانه اطلاعاتی و دو تای دیگر پیمانه افزونه است که قادر به تشخیص دو خطا و تصحیح یک خطا هستند و نوع مجموعه پیمانه به‌صورت $\{2^n+1, 2^n-1, 2^{2n}+1, 2^{2n}\}$ استفاده شده است. این مقاله به شرح زیر سازمان‌دهی شده است: در بخش ۲، مبانی مدل محاسباتی DNA و سیستم‌های RNS ، RRNS ارائه شده است. در بخش ۳، مجموعه پیمانه پیشنهادی و الگوریتم تشخیص و تصحیح خطا بیان شده است. بخش ۴، توابع و الگوریتم‌های جدید محاسبات DNA با تشخیص و تصحیح خطا، برای مدل ادلمن-لیپتون ارائه می‌شود. در نهایت، ارزیابی کار پیشنهادی و بحث و نتیجه‌گیری تکمیلی در بخش ۵ و۶ بیان شده‌اند.

۲- پیش‌زمینه

در این بخش، مدل ادلمن-لیپتون به عنوان مدل محاسباتی در DNA و ریاضیات مبتنی بر RNS معرفی شده‌اند.

۲-۱- اصول محاسبات DNA

یک رشته از DNA به عنوان یک رشته از چهار نوکلئوتید پایه مختلف تعریف می‌شود. چندین مدل برای محاسبات DNA پیشنهاد شده است[۱۲]. لوله آزمایش T مجموعه‌ای از توالی‌های DNA بر اساس حروف الفبای (A, G, C, T) است. برخی از عملیات اساسی DNA پیشنهاد شده توسط آدلمن و لیپتون به صورت زیر انجام می شوند [۱۰]:

ادغام (Merge): با دادن دو لوله آزمایش T_1 و T_2 ، عمل $\text{Merge}(T_1, T_2)$ الحاق $T_1 \cup T_2$ را در T_1 ذخیره می‌کند.

کپی (Copy): با دادن لوله آزمایش T_1 ، عمل $\text{Copy}(T_1, T_2)$ یک لوله آزمایش T_2 را با همان محتویات T_1 تولید می‌کند.

کشف (Detect): با دادن لوله آزمایش T_2، خروجی عمل $Detect(T)$ مقدار yes هست اگر T شامل حداقل یک رشته DNA باشد در غیر اینصورت مقدار خروجی برابر no می‌باشد.

[1] Adleman-Lipton

جداسازی (Separation): با دادن لوله آزمایش T_1 و یک مجموعه رشته‌های X، $\text{Separation}(T_1, X, T_2)$ همه تک رشته‌ها که شامل زیررشته X باشد را، از T_1حذف می‌کند و یک لوله آزمایش T_2 که شامل رشته‌های حذف شده باشد، تولید می‌کند.

انتخاب (Selection): با دادن لوله آزمایش T_1 و یک عدد صحیح L، $\text{Selection}(T_1, L, T_2)$ تمام رشته‌های درون لوله آزمایش T_1 که طولشان کمتر یا مساوی L باشد را حذف کرده و درون لوله آزمایش T_2 قرار می‌دهد.

شکافتن (Cleavage): با دادن لوله آزمایش T و یک رشته با دو سمبل $\sigma_0\sigma_1$، $\text{Cleavage}(T, \sigma_0\sigma_1)$ هر رشته دوتایی درون T که شامل $\left[\frac{\sigma_0\sigma_1}{\overline{\sigma_0\sigma_1}}\right]$ است را به دو تا رشته دوتایی به‌صورت زیر برش می‌دهد.

$$\begin{bmatrix}\alpha_0\sigma_0\sigma_1\beta_0\\ \alpha_1\overline{\sigma_0\sigma_1}\beta_1\end{bmatrix} \Rightarrow \begin{bmatrix}\alpha_0\sigma_0\\ \alpha_1\overline{\sigma}_0\end{bmatrix}, \begin{bmatrix}\sigma_1\beta_0\\ \overline{\sigma}_1\beta_1\end{bmatrix} \qquad (۱)$$

و فرض بر این است که این عملگر روی الفبای $\sum$ عمل می‌کند.

ترکیب یا ساختن (Annealing): با دادن لوله آزمایش T، $\text{Annealing}(T)$ همه دو رشته‌ای‌های ممکن از ترکیب تک رشته‌هایی که درون T هستند را می‌سازد و این دو رشته‌ای‌های جدید در همان لوله آزمایش T ذخیره می‌شوند.

تفکیک‌کردن (Denaturation): با دادن لوله آزمایش T، $\text{Denaturation}(T)$ هر دو رشته‌ای درون T را به دو تک رشته تبدیل می‌کند.

خالی کردن $\text{Empty}(T)$: با دادن لوله‌آزمایش T، $\text{Empty}(T)$ لوله Tرا خالی می‌کند.

۹ عملگر معرفی شده در بالا با تعداد ثابتی از مراحل بیولوژیکی برای رشته‌های DNA پیاده‌سازی شده‌اند. در اینجا فرض می‌شود که پیچیدگی هر عملگر $O(1)$ مرحله آزمایشگاهی است و پیچیدگی تقریبی از ترتیبی از عملگرها در نظر گرفته می‌شود [۱۳]. .

۲-۲- RNS و RRNS

انتشار رقم نقلی در سیستم عدد دودویی معمولی یک چالش اصلی جهت انجام عملیات ریاضی سریع است و به عنوان یک انگیزه کلیدی به کار برده شد؛ جهت اینکه این سیستم با سیستم عدد مانده‌ای جایگزین شود که در آن عملیات روی هر پیمانه به طور مستقل و موازی انجام می‌شود و مشکل انتشار رقم نقلی را ندارد [۱۵و۱۴].

سیستم اعداد مانده‌ای یک سیستم عددی غیروزنی هست که از مجموعه‌ای از اعداد صحیح مثبت پیمانه‌ها $(m_1, m_2, \dots, m_n)$ که دوبه‌دو نسبت به هم اول هستند تشکیل شده است یعنی $\gcd(m_i, m_j)=1$ برای $i \neq j$. یک عدد وزندار X می‌تواند به صورت $X = (x_1, x_2, \dots, x_n)$ نمایش داده می‌شود به‌طوری‌که:

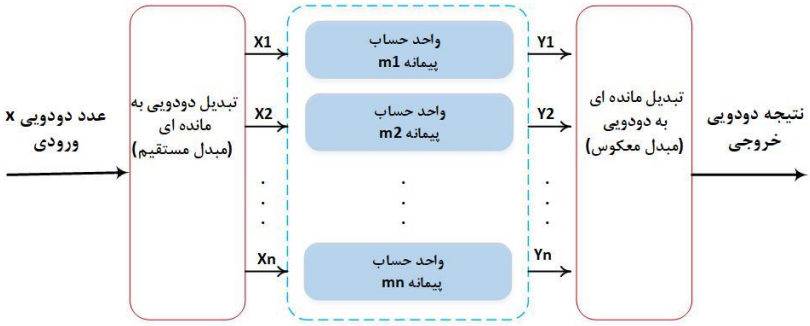
$$x_i = X \bmod m_i = |X|_{m_i} \quad , \quad 0 \leq x_i < m_i \qquad (۲)$$

چنین نمایشی برای هر عدد صحیح X در محدوده $[0, M-1]$ منحصربه‌فرد است، به‌طوری‌که M محدوده دینامیکی مجموعه پیمانه‌های $\{m_1, m_2, \dots, m_n\}$ است و برابر است با حاصل‌ضرب m_iها $(M = m_1, m_2, \dots, m_n)$ یعنی $M = \prod_{i=1}^{n} m_i$ [۱۶].

سیستم اعداد مانده‌ای یک سیستم عددی غیروزنی است که محاسبات موازی و سریع، کم‌مصرف و امن را پشتیبانی می‌کند. این سیستم از مجموعه‌ای از باقی‌مانده‌های تقسیم عدد بر پیمانه‌های مشخص، برای نمایش آن‌ها استفاده می‌کند. یکی از مهم‌ترین خواص سیستم اعداد مانده‌ای، انتقال محدود رقم نقلی در محاسبات می‌باشد که این ویژگی خاصیت ذاتی عدم انتشار رقم نقلی در عملیات حسابی نظیر جمع، تفریق و ضرب می‌باشد و محاسبه روی همه پیمانه‌ها می‌تواند به‌صورت هم‌زمان اجرا شود. در این سیستم عملیات به‌جای اینکه روی یک عدد بزرگ اجرا شود، روی چند عدد کوچک به‌صورت موازی انجام می‌شود [۱۸و۱۷].

یک مبدل مستقیم، شامل تبدیل کننده‌های مستقل برای هر پیمانه از مجموعه پیمانه می‌باشد که یک عدد دودویی صحیح وزندار را به عدد سیستم اعداد مانده‌ای معادلش تبدیل می‌کند. محاسبه باقی‌مانده در هر پیمانه به صورت جداگانه انجام می‌شود. این مبدل نسبت به مبدل معکوس، دارای طراحی آسان و پیچیدگی کمتری می باشد. بعد از اینکه نمایش مانده‌ای از عدد وزن‌دار توسط مبدل مستقیم به‌دست آمد، می‌توان عملیات حسابی مانند جمع، ضرب و تفریق را بر روی هر کدام از پیمانه‌ها به‌صورت مجزا و موازی انجام داد. نتیجه به‌دست‌آمده به یک مبدل معکوس انتقال می‌یابد تا به معادل دودویی تبدیل شود. عملیات حسابی در سیستم اعداد مانده‌ای بسیار سریع‌تر و ساده‌تر است. به دلیل این که باقی‌مانده‌ها نسبت به عدد اصلی کوچک‌تر هستند و به‌صورت مستقل انجام می‌شوند. در واقع سرعت انجام عملیات حسابی هنگامی که پیمانه‌ها کوچک باشند، بسیار بالا خواهد بود. هر پیمانه در مجموعه پیمانه پردازنده حسابی خودش را دارد که شامل پیمانه‌های جمع، تفریق و ضرب می‌باشد. بنابراین سیستم اعداد مانده‌ای شامل سه بخش مبدل مستقیم[1]، کانال پیمانه ای یا واحد حساب و مبدل معکوس[2] می‌باشد [۱۹].

[1] forward conversion
[2] Reverse conversion

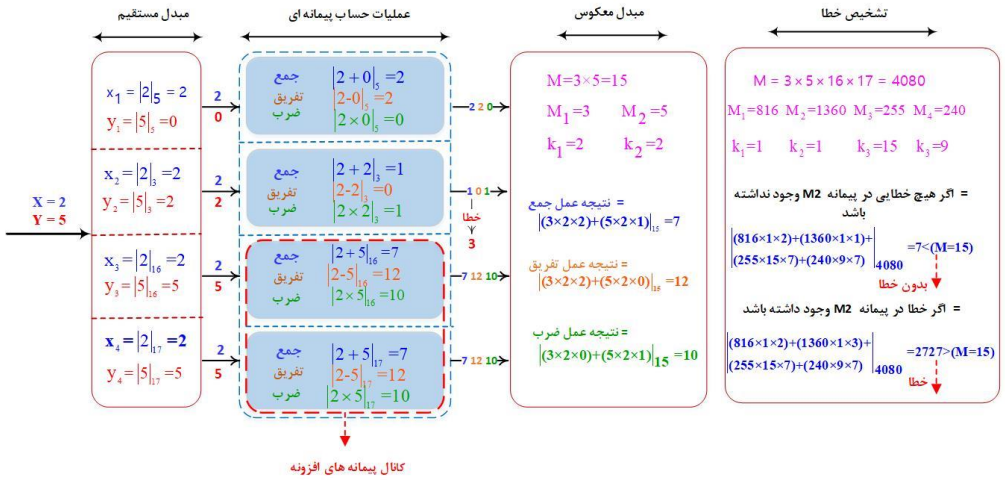


شکل ۱: بلوک دیاگرام یک سیستم نمونه RNS [۱۹]

در سیستم اعداد مانده‌ای، با اضافه کردن پیمانه‌های افزونه به مجموعه پیمانه‌های موجود می‌توان سیستمی داشت که دارای قابلیت تشخیص و تصحیح خطا باشد که به آن سیستم اعداد مانده‌ای افزونه یا RRNS گفته می‌شود. یک سیستم اعداد مانده‌ای افزونه، $n = k+r$ پیمانه اول $\{m_1, m_2, \dots, m_k, m_{k+1}, \dots, m_n\}$ و یک محدوده دینامیکی $M_N = \prod_{i=1}^{n} m_i$ دارد. به پیمانه‌های اضافه‌شده $m_{k+}m_{k+1}, \dots, m_{k+r}$ پیمانه افزونه گفته می‌شود و در مقابل به پیمانه‌های $m_1, m_2, \dots, m_k$ پیمانه اطلاعات گفته می‌شود. به فاصله $[0, M_k - 1]$ مربوط به k رقم مانده‌ای اطلاعات از مجموعه n تائی محدوده قانونی و به فاصله $[M_k, M_N - 1]$ با توجه به r رقم مانده‌ای افزونه ، محدوده غیرقانونی گفته می‌شود. محدوده قانونی، محدوده محاسباتی مفید سیستم عددی را نشان می‌دهد؛ درحالی‌که محدوده غیرقانونی، برای تشخیص خطا و سرریز مفید است. تعداد خطاهای قابل کشف و تصحیح بستگی به تعداد پیمانه‌های افزونه اضافه‌شده دارد. با r پیمانه افزونه، یک سیستم اعداد مانده‌ای افزونه قادر به تشخیص r و تصحیح $\lfloor r/2 \rfloor$ خطا می‌باشد. در نمایش باقیمانده‌ای همان‌طور که عملیات ریاضی پیمانه‌ای روی عملوندها در حال انجام هست؛ RRNS می‌تواند خطاهای حاصل از عملیات ریاضی را اصلاح کند. همچنین در RRNS باقی‌مانده‌ها مستقل از یکدیگر هستند، که باعث می‌شود خطای باقی‌مانده در یک کانال پیمانه به کانال‌های دیگر منتقل نشود. بنابراین، خطاهای وارد شده به رقم باقی‌مانده تنها اثر محلی دارند[۷].

شکل ۲، یک مثال عددی از یک RNS با پیمانه‌های (۳و۵) و یک RRNS با پیمانه‌های (۱۷و۱۶و۳و۵) را نشان می‌دهد. ۳ و ۵ پیمانه اطلاعات و ۱۶و ۱۷ پیمانه افزونه می‌باشند. ۲ و ۵ هم دو عدد صحیح هستند که باقی‌مانده‌های آن‌ها با مبدل مستقیم به صورت (۲،۲،۲،۲) و (۰،۲،۵،۵) به ترتیب می‌باشند. عملیات ریاضی روی باقی‌مانده‌های این دو عدد در هر پیمانه به صورت مستقل انجام می‌شود. نتایج این عملیات تبدیل به عدد صحیح توسط الگوریتم مبدل معکوس می‌شود. در این مثال از CRT جهت انجام مبدل معکوس استفاده می‌شود. مراحل محاسباتی برای تبدل معکوس نتایج

جمع، تفریق و ضرب برای سیستم RNS در این شکل نشان داده شده است. ارقام افزونه در RRNS می‌توانند برای تشخیص خطا استفاده شود. فرض کنید که یک اشکال در عمل جمع اتفاق می‌افتد و رقم مانده‌ای حاصل جمع در پیمانه دوم از ۱ به ۳ تغییر یابد. با در نظر گرفتن پیمانه افزونه به CRT عدد ۲۷۲۷ از مبدل معکوس به‌دست می آید که خارج از محدوده قانونی۱۵ =M (حاصل‌ضرب پیمانه‌های اطلاعات) می‌باشد. به این طریق خطا تشخیص داده شد؛ چون این RRNS فقط یک پیمانه افزونه دارد بنابراین فقط یک خطا را می‌تواند تشخیص دهد. برای تصحیح این خطا RRNS باید حداقل دو پیمانه افزونه داشته باشد [۲۰و۷].



شکل۲ : مثالی از محاسبات در RNS و RRNS

۲-۳- DNA و RNS

۱) *نمایش اعداد باینری و صحیح مانده ای با رشته های DNA در مدل ادلمن-لیپتون*

در [۱۳] یک نمایش DNA از اعداد باینری ارائه شده است. به‌طوری‌که یک رشته تکی DNA برای نمایش یک بیت استفاده می‌شود. نمایش m عدد باینری n بیتی مشابه [۸] در اینجا نشان داده شده است. الفبای $\sum$ و نمایش عدد به صورت زیر تعریف می‌شود:

$$\sum = \left\{ \begin{matrix} A_0, A_1, \dots, A_{m-1}, B_0, B_1, \dots, B_{n-1}, \\ C_0, C_1, E_0, E_1, D_0, D_1, 1,0, \bar{A}_0, \bar{A}_1, \dots, \bar{A}_{m-1}, \dots, \\ \bar{B}_0, \bar{B}_1, \dots, \bar{B}_{n-1}, \bar{C}_0, \bar{C}_1, \bar{E}_0, \bar{E}_1, \bar{D}_0, \bar{D}_1, \bar{1}, \bar{0}, \# \end{matrix} \right\} \quad (۳)$$

در (۳)، A آدرس عدد و B موقعیت بیت را مشخص می‌کنند و $C_0, C_1, E_0, E_1, D_0, D_1$ در عملیات برش به‌کار برده می‌شوند. سمبل های ۰ و۱ مشخص‌کننده مقدار هر بیت می‌باشند و "#" برای عملیات جداسازی استفاده می‌شود. با استفاده از الفبای بالا مقدار یک بیت، به وسیله یک رشته تکی به صورت زیر نمایش داده می‌شود که i آدرس و j موقعیت بیت را مشخص می‌کنند:

$$S_{ij} = D_1 A_i B_j C_0 C_1 V_{i,j} D_0. \quad (۴)$$

به‌طوری‌که $V_{ij} = 0$ اگر مقدار بیت صفر باشد، در غیر اینصورت $V_{ij} = 1$.

در [۱۲و۹] نمایش‌های DNA، m عدد صحیح n بیتی در RRNS به صورت زیر ارائه شده‌اند.

$$\sum = \{A_i, B_j, m_l, C_0, C_1, E_0, E_1, D_0, D_1, 1, 0, \# \mid \ 0 \le i \le m-1, 0 \le j \le n-1, 1 \le l \le 4\} \quad (۵)$$

هر سمبول در $\sum$ یک بخش از رشته تکی DNA را مشخص می‌کند. برای مجموعه پیمانه ۴ تایی $\{2^n+1, 2^n-1, 2^{2n}+1, 2^{2n}\}$ ، هر عدد باقی‌مانده در محدوده دینامیکی به وسیله یک مجموعه چهارتایی به صورت $\{S_{ijl}(V) = (A_i, B_j, m_l, V)\}$ نمایش داده می شود؛ که $V \in \{0,1\}$ و $S_{ijl}(V)$ یک رشته حافظه نامیده می‌شود. در این نمایش، i اندیس عدد در محدوده دینامیکی (کدامین عدد)، l موقعیت رقم باقی‌مانده و j موقعیت بیت، و V مشخص‌کننده مقدار بیت ۰ یا ۱ می‌باشد و سمبول‌های دیگر در مجموعه الفبای $\sum$ برای عملیات Cleavage و Separation استفاده می‌شوند. بنابراین یک رشته حافظه بر اساس الفبای $\sum$ به صورت زیر ساخته می‌شود:

$$S_{ijl}(V) = D_1 B_j E_0 E_1 A_i m_l C_0 C_1 V D_0. \quad (۶)$$

برای مثال، عدد صحیح ۵ در سیستم اعداد مانده‌ای با نمایش $(0,2,5,5)$ به وسیله ۴ مجموعه رشته‌های تکی DNA به‌صورت زیر نمایش داده می‌شود که در چهار لوله زیر ذخیره می‌شوند.

$$\begin{aligned} 0 &\rightarrow T_1 = \{S_{521}(0), S_{511}(0), S_{501}(0)\}, \\ 2 &\rightarrow T_2 = \{S_{522}(0), S_{512}(1), S_{502}(0)\}, \\ 5 &\rightarrow T_3 = \{S_{523}(1), S_{513}(0), S_{503}(1)\}, \\ 5 &\rightarrow T_4 = \{S_{524}(1), S_{514}(0), S_{504}(1)\} \end{aligned}$$

۲) عملیات اولیه مبتنی بر DNA

با استفاده از عملیات اساسی DNA چندین تابع برای ریاضیات مبتنی بر DNA و RNS تعریف شده‌اند. با توجه به مدل ادلمن- لیپتون توابع پایه‌ای به صورت زیر هستند:

تابع ValueAssignment (T_{input} , T_{output}) [۱۰و۸] : یک مقدار جدید یکسان $V(\epsilon \{0,1\})$ را به بیت مورد نظر در همه رشته‌های حافظه در لوله ورودی T_{input} تخصیص می‌دهد و نتیجه در لوله T_{output} ذخیره می‌شود.

تابع Leftshift (T_{input}) [۱۰]: این تابع برای انتقال اعداد استفاده می‌شود. برای یک عدد دودویی $a = (a_{s-1}, \dots, a_0)$ $\sum_{j=0}^{s-1} a_j 2^j$ ، الگوریتم شیفت به چپ مقدار $a' = \sum_{j=1}^{s-1} a_{j-1} 2^j = (a_{s-2}, \dots, a_0, 0)$ را بر می‌گرداند. در ابتدا مقدار عدد دودویی a در یک رشته حافظه درون تیوب ورودی به نام T_{input} ذخیره می‌شود، بعد از انجام عملیات الگوریتم تیوب T_{input} رشته حافظه‌هایی را ذخیره می‌کند که نتیجه a' را نشان می‌دهند.

تابع LogicOperation (T_{input}, L, T_{output}) [۲۱]: این تابع عملیات منطقی را بین دو عدد باینری انجام می‌دهد. این تابع عمل منطقی که در لوله L تعریف شده است را روی جفت رشته‌ای که در لوله T_{input} قرار دارند اعمال می‌کند و نتیجه عملیات منطقی را در لوله T_{output} ذخیره می‌کند.

تابع Binaryadd (T_1, T_2, T_{sum}) [۱۰] : دو عدد باینری که در رشته حافظه T_1, T_2 ذخیره شده‌اند را به‌صورت دودویی با هم جمع کرده و نتیجه عمل جمع را در رشته حافظه T_{sum} ذخیره می‌کند.

تابع EX_OR_ multi_operand (T_{input}) [۲۲] : عملیات EXOR را روی n عدد باینری m بیتی انجام می‌دهد که لوله T_{input} حاوی n عدد ورودی m بیتی که در رشته‌های حافظه ذخیره شده‌اند .

تابع AND_multi_operand (T_{input}) [۲۲] : همانند تابع EXOR عملیات AND را روی n عدد m بیتی انجام می‌دهد.

تابع RNSAdd ($T_{m-1}^x, \dots, T_0^x, T_{m-1}^y, \dots, T_0^y$) [۲۲] : این تابع دو عدد مانده‌ای با مجموعه پیمانه m تایی را با هم جمع می‌کند.

۳- مجموعه پیمانه پیشنهادی و تشخیص و تصحیح خطا

۱) انتخاب مجموعه پیمانه $\{2^n+1, 2^n-1, 2^{2n}+1, 2^{2n}\}$

انتخاب مناسب مجموعه پیمانه، نقش مهمی در طراحی کارامد سیستم اعداد مانده‌ای ایفا می‌کند. از پارامترهای مهم برای به‌دست آوردن محدوده دینامیک بزرگ در سیستم اعداد مانده‌ای انتخاب مجموعه پیمانه و تعداد آن‌هاست. از طرفی

انتخاب پیمانه و تعداد آن‌ها باید به گونه‌ای باشد که در ضمن فراهم آوردن محدوده نمایش بزرگ دارای سرعت محاسبات و همچنین پیچیدگی سخت‌افزاری مناسبی باشد. علاوه بر این هزینه و عملکرد پیمانه‌ها به شکل و تعداد پیمانه‌ها در مجموعه نیز بستگی دارد [۲۳]. مجموعه پیمانه‌های خوش‌فرم و متعادل زیادی وجود دارند که برای انجام محاسبات مناسب می‌باشند ولی به دلیل ساختار مبدل معکوس ناکارامد دارای هزینه زیاد و تاخیر در انجام عملیات هستند. از میان مجموعه پیمانه‌های موجود که دارای قابلیت‌های موازی سازی، تحمل خطا، محدوده دینامیکی بالا و پیاده سازی سخت افزاری ساده که سازگار با ویژگی های DNA باشد و همچنین قابلیت تصحیح خطا داشته باشد، مجموعه پیمانه ۴ تایی $\{2^n+1, 2^n-1, 2^{2n}+1, 2^{2n}\}$ برای کار ارائه شده در اینجا انتخاب شد [۲۴] که محدوده دینامیکی بالایی با ساختار مبدل معکوس ساده دارد که این ساختار را می‌توان به طور موثر با DNA پیاده سازی کرد. این مجموعه پیمانه 6n بیتی شامل پیمانه‌های ساده و خوش‌فرم که دارای خصوصیات سرعت بالا و طراحی مبدل معکوس با هزینه کم می باشد. در این مجموعه پیمانه از مبدل معکوس CRT1 استفاده می‌شود که باعث عملکرد بالا می‌شود. این مبدل تاخیر تبدیل کم و نیازمندی سخت افزاری کمتری را دارد.

۲) تشخیص و تصحیح خطا بر اساس مجموعه پیمانه $\{2^n+1, 2^n-1, 2^{2n}+1, 2^{2n}\}$

با توجه به مجموعه پیمانه چهارتایی $\{2^n+1, 2^n-1, 2^{2n}+1, 2^{2n}\}$ که دو پیمانه اول، جزء پیمانه اطلاعاتی و دو پیمانه دوم، جزء پیمانه افزونه می‌باشد، می توان تشخیص دو خطا و تصحیح یک خطا را انجام داد. جهت تشخیص و تصحیح خطا در اینجا از الگوریتم [۲۵] استفاده شده است. که بر اساس الگوریتم بررسی سازگاری[1] و گسترش پایه BEX[2] عمل می‌کند که نیاز به سخت افزار زیادی نخواهد بود. این الگوریتم برای یک سیستم با دو پیمانه اطلاعاتی (m_1, m_2) و دو پیمانه افزونه (m_3, m_4)، برای هر عدد $X(< M=m_1m_2m_3m_4)$ به‌صورت زیر است [۲۵].

مرحله ۱: مجموعه پیمانه $M = \{m1, m2, m3, m4\}$ و $X = (x1 , x2 , x3 , x4)$ به عنوان باقی‌مانده‌های انتقال یافته در نظر بگیرید که باید همان باقی‌مانده‌های اصلی X در صورت رخ ندادن خطا باشد.

مرحله ۲: عدد وزندار X'' باید با توجه به پیمانه‌های اطلاعاتی $\{m1, m2\}$ و باقی‌مانده‌های متناظر $(x1 , x2)$ با استفاده از CRT1 به صورت $x'' = x'_1 + m_1|(x'_2 - x'_1)k_1|_{m_2}$ محاسبه شود که در آن k_1 معکوس ضربی m_1 به پیمانه m_2 است.

مرحله ۳: $\Delta_{m_i} = X'' - x'_i$ برای i=3,4 یعنی برای پیمانه‌های افزونه $\{m3, m4\}$ محاسبه شود.

[1] checking consistency
[2] Base-Extension

مرحله ۴: اگر هر دو Δ برابر با صفر باشند، هیچ خطایی وجود ندارد و همه باقی‌مانده‌ها صحیح هستند. در غیر این صورت، اگر تنها یک Δ غیر صفر هست، یک خطا در باقی‌مانده افزونه وجود دارد و به مرحله ۵ برو. اگر هر دو Δ غیر صفر باشند، یک خطا در بخش اطلاعاتی وجود دارد و سپس به مرحله ۶ برو.

مرحله ۵: اگر Δ_{m_i} غیر صفر هست، پس x_i' صحیح نیست، که با جای‌گذاری آن با $|X''|_{m_i}$ تصحیح می‌شود.

مرحله ۶: پیمانه‌های افزونه را با پیمانه‌های اطلاعاتی جابه‌جا کن سپس $x'' = x_3' + m_3|(x_4' - x_3')k_2|_{m_4}$ و $\Delta_{m_i} = |X''|_{m_i} - x_i'$ را برای i=3,4 محاسبه کن. اگر هر دو Δ غیر صفر هستند یعنی چندین خطا وجود دارد، در غیر این صورت اگر فقط یک Δ غیر صفر باشد، سپس x_i' با $|X''|_{m_i}$ جایگزین شود.

پیاده‌سازی این الگوریتم با دو مثال به صورت زیر شرح داده می‌شود:

اولین مثال تصحیح یک خطا در پیمانه افزونه را نشان می‌دهد، در حالی که دومین مثال وجود خطا در پیمانه اطلاعاتی را نشان می‌دهد. عدد X=10 را در نظر بگیرید که نمایش با توجه به مجموعه پیمانه (۱۷، ۱۶، ۳، ۵) به صورت (۱۰، ۱۰، ۱، ۰) می‌باشد در حالی که ۳ و ۵ پیمانه اطلاعاتی و ۱۶ و ۱۷ پیمانه افزونه هستند.

در مثال اول فرض کنید که r_3 از مقدار ۱۰ به مقدار ۳ تغییر کرده است (یعنی خطا دارد)، به طوری که کد دریافت‌شده به صورت $x'= (0, 1, \underline{3}, 10)$ است. مراحل انجام این الگوریتم به صورت زیر می‌باشد:

مرحله ۲: $x'' = 0 + 5 \times |2(1-0)|_3 = 10$

مرحله ۳: $\Delta_{16}= 10 - 3 = 7 \neq 0 \quad , \Delta_{17}= 10 - 10 = 0$

مرحله ۴: با توجه به اینکه $\Delta_{17}= 0 \ , \Delta_{16}\neq 0$ ، بنابراین یک خطا در بخش افزونه وجود دارد.

مرحله ۵: بنابراین x_3' صحیح نمی باشد و دارای خطا هست. برای تصحیح x_3' با $|10|_{16}$ جایگزین می‌شود، به این صورت $x'= (0, 1, \underline{10}, 10)$.

در مثال دوم، فرض کنید که r_1 از مقدار ۰، به مقدار ۲ تغییر کرده اســت (یعنی خطا دارد). به طوری که کد دریافت شده به صورت $x'= (\underline{2}, 1, 10, 10)$ است. مراحل انجام این الگوریتم به صورت زیر می‌باشد:

مرحله ۲: $x'' = 2 + 5 \times |2(1-2)|_3 = 7$

مرحله ۳: $\Delta_{16}= 7 - 10 \neq 0 \quad , \quad \Delta_{17}= 7 - 10 \neq 0$

مرحله ۴: با توجه به اینکه $\Delta_{17}\neq 0 \ , \Delta_{16}\neq 0$ ، بنابراین یک خطا در بخش اطلاعاتی وجود دارد.

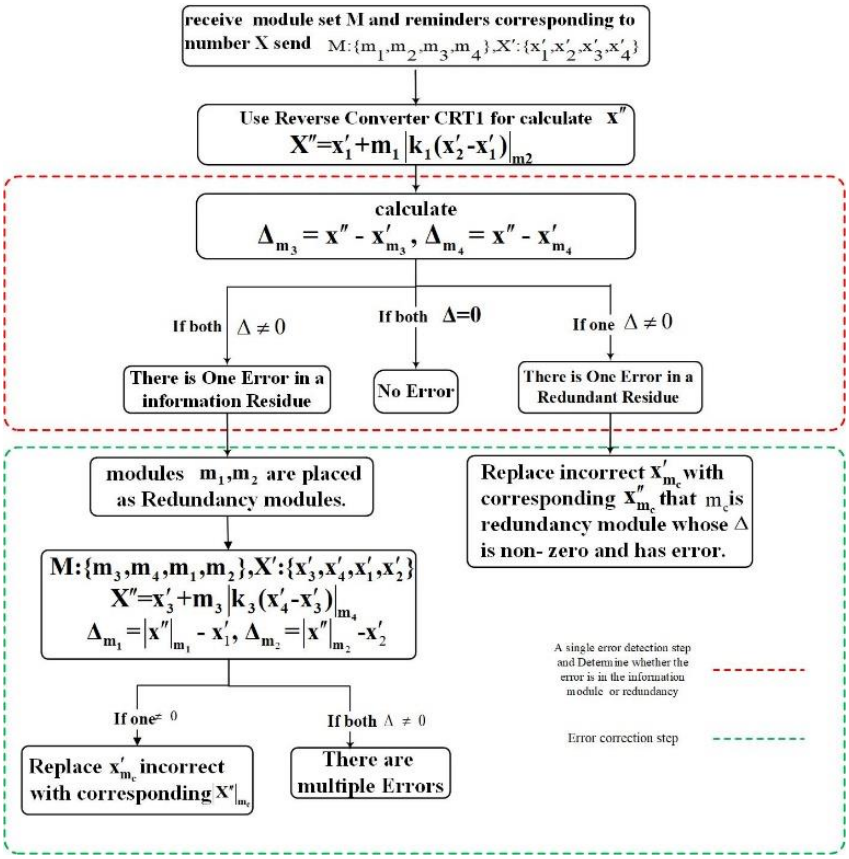
مرحله ۶: $x'= (10, 10, \underline{2}, 1)$ and $M= (16, 17, 5, 3)$

$x'' = 10 + 16 \times |16(10-10)|_{17} = 10$

$|10|_5 = 0,\ \ |10|_3 = 1$

$\Delta_5 = 0 - 2 \neq 0,\ \ \Delta_3 = 1 - 1 = 0$

با توجه به اینکه $\Delta_3 = 0$ و $\Delta_5 \neq 0$، بنابراین یک خطا در بخش افزونه وجود دارد؛ به‌طوری‌که x'_1 با مقدار $|10|_5 = 0$ جایگزین می‌شود و بدین ترتیب X'= (0, 1, 10, 10). شکل ۳، الگوریتم تشخیص و تصحیح یک خطا را نشان می‌دهد.



شکل ۳: الگوریتم تشخیص خطا و تصحیح یک خطا

۴- الگوریتم‌های حسابی DNA پیشنهادی بر اساس مدل ادلمن-لیپتون با تشخیص دو خطا و تصحیح یک خطا

همان‌طور که قبلاً گفته شد، یکی از ویژگی‌های سیستم اعداد مانده‌ای، عدم انتشار رقم نقلی است که عملیات محاسبات مولکولی را ساده می‌کند و از موازی‌سازی در محاسبات مولکولی و کارایی آن به طور مؤثر استفاده می‌کند. همچنین با توجه به خاصیت تشخیص و تصحیح خطا در سیستم اعداد مانده‌ای، در اینجا از آن برای تشخیص دو خطا و تصحیح یک

خطا در محاسبات مولکولی استفاده شده است. در این بخش الگوریتم‌های پیشنهادی برای انجام عملیات ریاضی DNA با تشخیص و تصحیح خطا بر اساس سیستم اعداد مانده‌ای افزونه، برای مدل ادلمن- لیپتون ارائه و پیاده‌سازی شده است. در بخش اول یک الگوریتم عملیات ریاضی به نام جمع دو عدد مانده‌ای در سیستم DNA ارائه می‌شود و در بخش دوم الگوریتم‌های جدید، برای تحقق توابع مورد نیاز برای پیاده سازی حساب DNA با اعداد مانده‌ای ارائه شده است. برای تحقق توابع موردنظر، برای پیاده‌سازی محاسبات مولکولی با سیستم اعداد مانده‌ای افزونه پیاده‌سازی مدل ادلمن – لیپتون به زبان سی پلاس پلاس در محیط ویژوال استودیو انجام شده است.

۱) جمع ریاضی دو عدد در سیستم اعداد مانده‌ای

در این روش ابتدا دو عدد باینری عادی که در تیوب های T_x و T_y قرار دارند توسط توابع Forward به اعدادی در مجموعه سیستم اعداد مانده‌ای تبدیل می‌شوند که اعداد مانده‌ای عدد اول در تیوب های $T_{x'_1}$ تا $T_{x'_4}$ و اعداد مانده‌ای عدد دوم در تیوب های $T_{y'_1}$ تا $T_{y'_4}$ قرار می‌گیرند. سپس عملیات جمع اعداد مانده‌ای، با توجه به پیمانه های $\{2^n+1, 2^n-1, 2^{2n}+1, 2^{2n}\}$ روی آنها انجام می‌شود. در ادامه نتیجه عمل جمع توسط تابع تشخیص و تصحیح خطا بررسی می‌شود. این روش در الگوریتم ۱ نشان داده شده است. عملیات حسابی می‌تواند هر عملیاتی باشد که هدف در اینجا تشخیص و تصحیح خطاها می‌باشد.

***Algorithm 1*_" the proposed DNA based with *Adleman-Lipton* "_ addition two RNS numbers**

```
Procedure   addition_two numbers RNS (input: Tx, Ty, Tm1, Tm2 , Tm3, Tm4, n)
{
   Forward (Tx, Tm1, Tm2 , Tm3, Tm4, n, Tx'1 , Tx'2  , Tx'3  , Tx'4 );
   Forward (Ty, Tm1, Tm2 , Tm3, Tm4, n, Ty'1 , Ty'2  , Ty'3  , Ty'4 );
   Add_mod2np1 (Tx'1 , Ty'1 , n, Tre1 , Toverflow);
   Add_mod2nm1 (Tx'2 , Ty'2 , n, Tre2 , Toverflow);
   Add_mod2np1 (Tx'3 , Ty'3 , 2n, Tre3 , Toverflow);
   Add_mod2n (Tx'4 , Ty'4 , 2n, Tre4 , Toverflow);
   Error_Detection_Correction_DNA (Tre1, Tre2, Tre3, Tre4, Tm1, Tm2 , Tm3, Tm4 );
}
```

۲) توابع جدید در مدل ادلمن – لیپتون

در این بخش توابع جدید برای عملیات جمع و تشخیص و تصحیح خطا در محاسبات مولکولی بر اساس سیستم اعداد مانده‌ای افزونه در مدل ادلمن – لیپتون که در قسمت قبل معرفی شد، تعریف و پیاده‌سازی شده‌اند.

تابع جمع و تفریق پیمانه‌ای در مدل ادلمن – لیپتون

عملیات جمع و تفریق دو عمل اصلی در محاسبات سیستم اعداد مانده‌ای هستند. این عملیات نه‌تنها در محاسبات ریاضی سیستم اعداد مانده‌ای موردنیاز هستند؛ بلکه در مبدل مستقیم و معکوس هم موردنیاز هستند. تابع عملیات جمع پیمانه‌ای، در الگوریتم ۲ برای پیمانه‌های $2^n-1, 2^n+1$ و 2^n نشان داده شده است. در این توابع ورودی اعداد مانده‌ای هستند و در T_x T_y, به عنوان ورودی تابع تعریف شده‌اند. همچنین ورودی n به عنوان طول اعداد تعریف شده است و حاصل عملیات جمع در خروجی T_{sum} ذخیره می‌شود. تابع Add_mod2np1 برای جمع مانده‌ای با توجه به پیمانه 2^n+1 ، تابع Add_mod2nm1 برای جمع مانده‌ای با توجه به پیمانه 2^n-1، تابع Add_mod2np1 برای جمع مانده‌ای، با توجه به پیمانه $2^{2n}+1$ با 2n بیت و تابع Add_mod2n برای جمع مانده‌ای با توجه به پیمانه 2^{2n} انجام می‌شود و برای عملیات تفریق پیمانه‌ای اگر حاصل تفریق دو عدد مانده‌ای بزرگ‌تر از صفر باشد، پس جواب مساوی X-Y خواهد بود؛ یعنی حاصل تفریق کمتر از مقدار پیمانه m باشد (البته با فرض اینکه که ورودی‌ها هر کدام کمتر از پیمانه یعنی m هستند)؛ ولی اگر حاصل تفریق دو عدد مانده‌ای منفی باشد، یعنی رقم نقلی یک شود فقط کافی است که پیمانه یک‌بار با حاصل تفریق جمع شود تا حاصل درست به‌دست آید [۱۹]. این تابع در الگوریتم ۳ نشان داده شده است. برای انجام عملیات جمع و تفریق پیمانه‌ای نیاز به توابع عملیات جمع و تفریق معمولی و باقی‌مانده پیمانه‌ای هست. بنابراین توابع calculate_remain_mod2nm1 (الگوریتم ۴) و calculate_remain_mod2np1 (الگوریتم ۵) برای پیدا کردن باقی‌مانده بر اساس پیمانه‌های 2^n-1 و $j2^n+1$ به ترتیب استفاده می‌شود.

Algorithm 2_" the proposed DNA based with Adleman-Lipton "_Modular addition

```
Procedure add_mod2nm1(input: Tx, Ty, n; output: Tsum) {
   Add(Tx, Ty, n+1, Ttmp, Toverflow);
   calculate_remain_mod2nm1(Ttmp, n, Tsum);
   Empty (Ttmp);
   Empty (Toverflow);}

Procedure add_mod2np1 (Tx, Ty, n, Tsum ) {
   Add (Tx, Ty, n+1,Ttmp,Toverflow);
   calculate_remain_mod2np1 (Ttmp, n, Tsum);
   Empty (Ttmp);
   Empty (Toverflow);}

Procedure add_mod2n (Tx, Ty, n, Tsum) {
   Add (Tx, Ty, n+1, Tsum, Toverflow);
   Empty (Toverflow) ;}
```

Algorithm 3_" the proposed DNA based with Adleman-Lipton"_ Modular subtraction

Procedure subtract_mod (input: T_x, T_y, T_m, n; output: T_{remain}) {

Subtract (T_x, T_y, n, T_{remain}, $T_{overflow}$);

If ($T_{overflow}$ not empty)

Subtract (T_x, T_y, n, T_{sub}, $T_{overflow}$);

Add (T_{sub}, T_m, n, T_{remain}, $T_{overflow}$);

End if

Empty ($T_{overflow}$) ;}

Algorithm 4_" the proposed DNA based with Adleman-Lipton"_ calculation remain moduli $2^n - 1$

Procedure calculate_remain_mod2nm1 (input:T_x, n; output: T_{remain})

{ // all bits of T_{one} is one

Add (T_x, T_{one}, n, T_{remain}, $T_{overflow}$);

Separation (T_x , {B_n}, T_{tmp});

Separation (T_{tmp} , {0}, T_{bon});

Separation (T_{tmp} , {1}, T_{boff});

If (T_{bon} is not empty or $T_{overflow}$ is not empty)

Empty ($T_{overflow}$);

Merge (T_x , T_{bon});

Merge (T_x , T_{boff});

Else

Empty (T_{remain});

Merge (T_x , T_{boff});

For i=0 to n-1 do

Separation (T_x , {B_i}, T_{remain});

End for;

End if;

}

Algorithm 5_" the proposed DNA based with Adleman-Lipton"_ calculation remain moduli $2^n + 1$

Procedure calculate_remain_mod2np1 input: T_x, n; output: T_{remain}) {

Separation (T_x , {B_n}, T_{tmp});

Separation (T_{tmp} , {0}, T_{bon});

Separation (T_{tmp} , {1}, T_{boff});

Empty (T_{tmp});

If (T_{boff} is not empty)

Merge (T_x , T_{boff});

Separation (T_x , {B_{n+1}}, T_{tmp});

Separation (T_{tmp} , {0}, T_{bon});

Separation (T_{tmp} , {1}, T_{boff});

If (T_{bon} is not empty)

Merge (T_x , T_{bon});

Subtract (T_x, T_{one}, n, T_{remain}, $T_{overflow}$);

Empty ($T_{overflow}$);

Else

Merge (T_x , T_{boff});

For i=0 to n-1 do

Separation (T_x , {B_i}, T_{remain});

End for;

End if;

Else

Merge (T_x , T_{bon});

Subtract (T_x, T_{one}, n, T_{remain}, $T_{overflow}$);

If ($T_{overflow}$ is not empty)

Empty (T_{remain});

For i=0 to n do

Separation (T_x , {B_i}, T_{remain});

End for;

Else

Empty ($T_{overflow}$);

End if ;}

Algorithm 6_" the proposed DNA based with Adleman-Lipton"_addition with or without carry

```
Procedure Add ( Tx , Ty , n , Tout, Toverflow) {
//Tcarry and Tout are zero
  Merge (Tnum, Tx);
  Merge (Tnum, Ty);
  For i=1 to n
     Separation (Tnum , {Bi}, Ttmp);
     Separation (Ttmp , {0}, Tbon);
     Separation (Ttmp , {1}, Tboff);
     If ( Tcarry is empty )
        If (Tboff not empty and Tbon not empty)//both bits are differnt
         Separation (Tout , {Bi}, Tout1);
         Valueassignment (Tout1 , 1);
         Merge (Tout , Tout1);
```

تابع جمع و تفریق دو عدد با رقم نقلی و بدون رقم نقلی در مدل ادلمن – لیپتون

این تابع دو عدد n بیتی دریافت می‌کند که توسط دو رشته DNA مختلف در لوله آزمایشTx ، Ty قرار دارند و نتیجه عملیات جمع در لوله آزمایش Tout ذخیره و سرریز در Toverflow ارائه می‌شود. عملیات جمع یا با بیت نقلی یا بدون بیت نقلی انجام می‌شود. این تابع در الگوریتم ۶ نشان داده شده است. برای تفریق، متمم دو، عدد دوم با عدد اول مشابه الگوریتم جمع انجام می‌شود که در الگوریتم ۷ نشان داده شده است.

Algorithm 6_" the proposed DNA based with Adleman-Lipton"_addition with or without carry

Else if (T_{boff} is empty) //both bits are one

Separation (T_{carry} , $\{B_i\}$, T_{carry1});

Valueassignment (T_{carry1} , 1);

Merge (T_{carry} , T_{carry1});

End if

End

Else

If (T_{boff} not empty and T_{bon} not empty)

Separation (T_{carry} , $\{B_i\}$, T_{carry1});

Valueassignment (T_{carry1} , 1);

Merge (T_{carry} , T_{carry1});

Elseif (T_{boff} is empty)

Separation (T_{out} , $\{B_i\}$, T_{out1});

Valueassignment (T_{out1} , 1);

Merge (T_{out} , T_{out1});

Separation (T_{carry} , $\{B_i\}$, T_{carry1});

Valueassignment (T_{carry1} , 1);

Merge (T_{carry} , T_{carry1});

Else

Separation (T_{out} , $\{B_i\}$, T_{out1});

Valueassignment (T_{out1} , 1);

Merge (T_{out} , T_{out1});

End if

End if

End if

End for

If (T_{carry} not empty)

Valueassignment ($T_{overflow}$, 1); End if }

Algorithm 7_" the proposed DNA based with Adleman-Lipton"_subtract

```
Procedure subtract ( Tx , Ty , n , Tout, Toverflow) {
    Not_operation (Ty);
    Add (Ty , 1 , n , Tout1, Toverflow);
    Add (Tx , Tout1 , n , Tout, Toverflow);
}
```

تابع مبدل مستقیم در مدل ادلمن – لیپتون

فرایند تبدیل اعداد از سیستم وزنی به سیستم مانده‌ای توسط مبدل مستقیم انجام می‌گیرد. عدد X وارد مبدل دودویی به مانده‌ای شده و محاسبه باقی‌مانده در هر پیمانه به‌صورت جداگانه انجام می‌شود. با توجه به مجموعه پیمانه $\{2^n+1,$ 2^n-1, 2^{2n}+1, $2^{2n}\}$ در این الگوریتم عدد باینری جهت تبدیل در T_{in} قرار دارد که به عنوان ورودی داده می شود و همچنین پیمانه‌های مورد نظر به عنوان ورودی در T_m به تابع داده می‌شوند و T_x ها خروجی تابع می‌باشد که اعداد مانده‌ای تبدیل شده را برمی‌گرداند. این تابع در الگوریتم ۸ تعریف و پیاده سازی شده است.

Algorithm 8_" the proposed DNA based with Adleman-Lipton"_ forward converter

Procedure forward_converter (input: T_{in}, $T_{m_1}, T_{m_2}, T_{m_3}, T_{m_4}, n$; output: $T_{x_1'}, T_{x_2'}, T_{x_3'}, T_{x_4'}$) {

//T_{in} is 6n bit

//calculate $|T_{in}|_{T_{m_1}}$, $m_1 = 2^n + 1$

For i=0 to n-1 do

Separation (T_{in} ,{B_i}, T_{f_0});

Separation (T_{in} ,{B_{i+n}}, T_{f_1});

Separation (T_{in} ,{B_{i+2n}}, T_{f_2});

Separation (T_{in} ,{B_{i+3n}}, T_{f_3});

End for;

Add ($T_{f_0}, T_{f_2}, n+1$, $T_{sum_1}, T_{overflow}$);

Add ($T_{f_1}, T_{f_3}, n+1$, $T_{sum_2}, T_{overflow}$);

Subtract_mod ($T_{sum_1}, T_{sum_2}, T_{m_1}, n+1$; $T_{x_1'}$);

Empty (T_{sum_1});

Empty (T_{sum_2});

//calculate $|T_{in}|_{T_{m_2}}$, $m_2 = 2^n - 1$

Add (T_{f_0}, T_{f_1}, $n+1$, $T_{sum_1}, T_{overflow}$);

Add ($T_{f_2}, T_{f_3}, n+1$, $T_{sum_2}, T_{overflow}$);

calculate_remain_mod2nm1 (T_{sum_1}, n, T_1);

calculate_remain_mod2nm1 (T_{sum_2}, n, T_2);

Add (T_1, T_2, n, $T_{sum}, T_{overflow}$);

calculate_remain_mod2nm1 (T_{sum}, n, $T_{x_2'}$);

//calculate $|T_{in}|_{T_{m_3}}$, $m_3 = 2^{2n} + 1$

For i=0 to 2n-1 do

Separation (T_{in} ,{B_i}, T_{f_0});

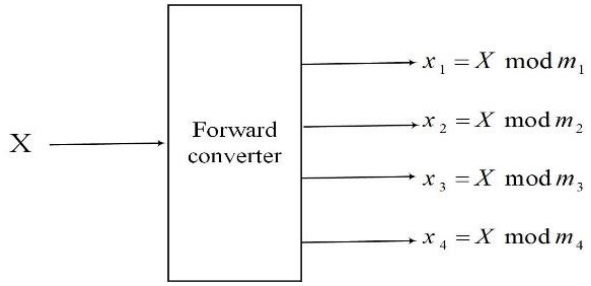
Separation (T_{in} ,{B_{i+2n}}, T_{f_1});

End for;

Subtract_mod ($T_{f_0}, T_{f_1}, T_{m_3}, 2n+1, T_{x_3'}$);

//calculate $|T_{in}|_{T_{m_4}}$, $m_4 = 2^{2n}$

Copy ($T_{f_0}, T_{x_4'}$) ;}



شکل ۴: مبدل مستقیم

محدوده دینامیکی در این مجموعه پیمانه برابر است با:

$$M=2^{n}-1 \times 2^{n}+1 \times 2^{2n} \times 2^{2n}+1 = 2^{4n}-1 \quad (۷)$$

که یک عدد 4n بیتی است یعنی:

$$X = X_{4n-1} \dots X_0 \quad (۸)$$

به عنوان مثال برای محدوده 4n بیتی داریم:

با توجه به اینکه $|2^{4n}|_{2^{2n}} = |2^{3n}|_{2^{2n}} = |2^{2n}|_{2^{2n}} = 0$ است، x_4 به صورت زیر به دست می آید[۱۹]:

$$x_4 = |X|_{2^{2n}} = \left| 2^{2n}(\underbrace{X_{4n-1} \dots X_{2n}}_{f_1}) + \underbrace{X_{2n-1} \dots X_0}_{f_0} \right|_{2^{2n}} = X_{2n-1} \dots X_0 \quad (۹)$$

از طرفی، با توجه به اینکه $|2^{3n}|_{2^n-1} = |2^{2n}|_{2^n-1} = |2^n|_{2^n-1} = 1$ است، x_2 به صورت زیر به دست می‌آید[۱۹]:

$$x_2 = |X|_{2^n-1} = \left| \begin{matrix} 2^{3n}\left(\underbrace{X_{4n-1} \dots X_{3n}}_{f_3}\right) + 2^{2n}\left(\underbrace{X_{3n-1} \dots X_{2n}}_{f_2}\right) \\ +2^{n}\left(\underbrace{X_{2n-1} \dots X_{n}}_{f_1}\right) + \underbrace{X_{n-1} \dots X_0}_{f_0} \end{matrix} \right|_{2^n-1} \quad (۱۰)$$

$$= \left| \begin{matrix} \underbrace{X_{4n-1} \dots X_{3n}}_{f_3} + \underbrace{X_{3n-1} \dots X_{2n}}_{f_2} + \\ \underbrace{X_{2n-1} \dots X_{n}}_{f_1} + \underbrace{X_{n-1} \dots X_0}_{f_0} \end{matrix} \right|_{2^n-1} = |f_3 + f_2 + f_1 + f_0|_{2^n-1}$$

با توجه به اینکه $|2^{2n}|_{2^n+1} = 1$ و $|2^{3n}|_{2^n+1} = |2^n|_{2^n+1} = -1$ است، x_1 به صورت زیر به دست می‌آید[۱۹]:

$$x_1 = |X|_{2^n+1} = \left| \begin{matrix} 2^{3n}\left(\underbrace{X_{4n-1} \dots X_{3n}}_{f_3}\right) + 2^{2n}\left(\underbrace{X_{3n-1} \dots X_{2n}}_{f_2}\right) + \\ 2^{n}\left(\underbrace{X_{2n-1} \dots X_{n}}_{f_1}\right) + \underbrace{X_{n-1} \dots X_0}_{f_0} \end{matrix} \right|_{2^n+1} \quad (۱۱)$$

$$= \left| \begin{matrix} -\underbrace{X_{4n-1} \dots X_{3n}}_{f_3} + \underbrace{X_{3n-1} \dots X_{2n}}_{f_2} - \\ \underbrace{X_{2n-1} \dots X_{n}}_{f_1} + \underbrace{X_{n-1} \dots X_0}_{f_0} \end{matrix} \right|_{2^n+1} = |-f_3 + f_2 - f_1 + f_0|_{2^n+1}$$

$$= |(f_0 + f_2) - (f_1 + f_3)|_{2^n+1}$$

در نهایت، با توجه به اینکه $|2^{2n}|_{2^{2n}+1} = -1$ است، x_3 به صورت زیر به دست می‌آید [۱۹]:

$$x_3 = |X|_{2^{2n}+1} = \left| 2^{2n} \left(\underbrace{X_{4n-1} \dots X_{2n}}_{f_1} \right) + \underbrace{X_{2n-1} \dots X_0}_{f_0} \right|_{2^{2n}+1} \quad (۱۲)$$
$$= \left| -\underbrace{X_{4n-1} \dots X_{2n}}_{f_1} + \underbrace{X_{2n-1} \dots X_0}_{f_0} \right|_{2^{2n}+1} = |f_0 - f_1|_{2^{2n}+1}$$

تابع مبدل معکوس در مدل ادلمن – لیپتون

این تابع دو عدد مانده‌ای را دریافت کرده و عدد صحیح را برمی‌گرداند. الگوریتم این تابع در الگوریتم‌های ۹ و ۱۰ نشان داده شده است. فرض کنید مجموعه مدول ۴ تایی $\{m_1, m_2, m_3, m_4\} = \{2^n+1, 2^n-1, 2^{2n}+1, 2^{2n}\}$ با مجموعه باقی‌مانده‌ای $\{x'_1, x'_2, x'_3, x'_4\}$ را داریم. عدد وزنی x'' با توجه به دو پیمانه اطلاعاتی اول به صورت زیر محاسبه می‌شود:

$$x'' = x'_1 + m_1|(x'_2 - x'_1)k_1|_{m_2} \quad (۱۳)$$

$$|k_1 \times m_1|_{m_2} = 1 \quad (۱۴)$$

معکوس ضربی از $2^n + 1$ به پیمانه $2^n - 1$ برابر با $k_1 = 2^{n-1}$ می‌باشد.

$$x'' = x'_1 + (2^n + 1)|(x'_2 - x'_1)2^{n-1}|_{2^n-1} \quad (۱۵)$$

قانون ۱: باقی‌مانده یک عدد مانده‌ای (v-) به پیمانه $2^p - 1$ برابر است با مکمل یک عدد v یعنی $|-v|_{2^p-1} = |\bar{v}|_{2^p-1}$ [۲۴].

قانون ۲: ضرب یک باقی‌مانده v در 2^p به پیمانه $2^n - 1$ برابر است با p بیت چرخشی به چپ عدد v یعنی $|2^p v|_{2^n-1}$ [۲۴].

با توجه به مجموعه پیمانه $\{2^n+1, 2^n-1, 2^{2n}+1, 2^{2n}\}$ و با توجه به باقی‌مانده‌های متناظر با عدد X به صورت $\{x'_1, x'_2, x'_3, x'_4\}$ ، نمایش بیتی آنها به صورت زیر می‌باشد:

$$x'' = x'_1 + (2^n + 1)|(x'_2 - x'_1)2^{n-1}|_{2^n-1} \quad (۱۵)$$

$$x'_1 = (\underbrace{x'_{1,n}\ x'_{1,n-1} \dots x'_{1,1}\ x'_{1,0}}_{n+1\ bits})_2 \quad (۱۶)$$

$$x'_2 = (\underbrace{x'_{2,n-1}\ x'_{2,n-2} \dots x'_{2,1}\ x'_{2,0}}_{n\ bits})_2 \quad (۱۷)$$

$$x_3' = (\underbrace{x_{3,2n}'\, x_{3,2n-1}'\, x_{3,2n-2}' \dots x_{3,1}'\, x_{3,0}'}_{2n+1\ bits})_2 \quad (۱۸)$$

$$x_4' = (\underbrace{x_{4,2n-1}'\, x_{4,2n-1}' \dots x_{4,1}'\, x_{4,0}'}_{2n\ bits})_2 \quad (۱۹)$$

اکنون، معادله (۱۵) می‌تواند به‌صورت زیر ساده شود [۲۵]:

$$x'' = x_1' + (2^n + 1) \times H \quad (۲۰)$$

$$H = |2^{n-1}(x_2' - x_1')|_{2^n-1} = |v_1 + v_2|_{2^n-1} \quad (۲۱)$$

$$v_1 = |2^{n-1}x_2'|_{2^n-1} = \left|2^{n-1}\left(x_{2,n-1}' \dots x_{2,0}'\right)\right|_{2^n-1} = \underbrace{x_{2,0}'x_{2,n-1}' \dots x_{2,2}'x_{2,1}'}_{n\ bits} \quad (۲۲)$$

$$v_2 = |-2^{n-1}x_1'|_{2^n-1} = \left|-2^{n-1}\left(x_{1,n}' \dots x_{1,0}'\right)\right|_{2^n-1} = \left|-2^{n-1}(\underbrace{x_{1,n}' \times 2^n + x_{1,n-1}' \dots x_{1,0}'}_{n+1\ bits})\right|_{2^{n-1}} \quad (۲۳)$$

$$v_{21} = \left|-2^{n-1}\left(x_{1,n-1}' \dots x_{1,0}'\right)\right|_{2^n-1} = \underbrace{\bar{x}_{1,0}'\bar{x}_{1,n-1}' \dots \bar{x}_{1,2}'\bar{x}_{1,1}'}_{n\ bits} \quad (۲۴)$$

$$v_{22} = \left|-2^{n-1} \times 2^n\left(0 \dots 00x_{1,n}'\right)\right|_{2^n-1} = \underbrace{01 \dots 11}_{n\ bits} \quad (۲۵)$$

$$v_2 = \begin{cases} v_{21} & if\ x_{1,n} = 0 \\ v_{22} & if\ x_{1,n} = 1 \end{cases} \quad (۲۶)$$

معکوس ضربی از $2^n + 1$ به پیمانه $2^n - 1$ برابر با $k_1 = 2^{n-1}$ می‌باشد.

$$x'' = x_1' + (2^n + 1)|(x_2' - x_1')2^{n-1}|_{2^n-1} \quad (۲۷)$$

به طور مشابه، عدد وزنی x'' با توجه به دو پیمانه افزونه آخر به صورت زیر محاسبه می‌شود:

$$x'' = x_3' + m_3|(x_4' - x_3')k_2|_{m_4} \quad (۲۸)$$

$$|k_2 \times m_3|_{m_4} = 1 \quad (۲۹)$$

معکوس ضربی از $2^{2n} + 1$ به پیمانه 2^{2n} برابر با $k_1 = 1$ می‌باشد، از این رو:

$$|k_2 \times (2^{2n} + 1)|_{2^{2n}} = |k_2 \times (1)|_{2^{2n}} = 1 \quad (۳۰)$$

بنابراین (۲۸) به‌صورت زیر می‌شود:

$$x'' = x_3' + (2^{2n} + 1)|x_4' - x_3'|_{2^{2n}} \quad (۳۱)$$

اکنون، برخی از ساده‌سازی‌های سطح بیت را در (۳۱) اعمال می‌کنیم تا یک پیاده‌سازی کارآمد با DNA داشته باشیم.

اول، می‌دانیم که $|A+B|_{2^p}$ برابر است با p بیت عمل جمع A و B با نادیده گرفتن بیت نقلی [A]. از این رو:

$$x'' = x'_3 + (2^{2n}+1)H \tag{۳۲}$$

$$\mathrm{H} = |x'_4 - x'_3|_{2^{2n}} = \left| \underbrace{x'_{4,2n-1}\, x'_{4,2n-1} \dots x'_{4,1}\, x'_{4,0}}_{2n\ bits} - \underbrace{x'_{3,2n-1}\, x'_{3,2n-2} \dots x'_{3,1}\, x'_{3,0}}_{2n+1\ bits} \right|_{2^{2n}} \tag{۳۳}$$

$$= \left| \underbrace{x'_{4,2n-1}\, x'_{4,2n-1} \dots x'_{4,1}\, x'_{4,0}}_{2n\ bits} - \underbrace{x'_{3,2n} \times 2^{2n} + x'_{3,2n-1}\, x'_{3,2n-2} \dots x'_{3,1}\, x'_{3,0}}_{2n+1\ bits} \right|_{2^{2n}}$$

این واضح است که $\left|x'_{3,2n} \times 2^{2n}\right|_{2^{2n}} = 0$ ، بنابراین:

$$\begin{aligned} \mathrm{H} &= \left| \underbrace{x'_{4,2n-1}\, x'_{4,2n-1} \dots x'_{4,1}\, x'_{4,0}}_{2n\ bits} - \underbrace{x'_{3,2n-1}\, x'_{3,2n-2} \dots x'_{3,1}\, x'_{3,0}}_{2n\ bits} \right|_{2^{2n}} \\ &= \left| \underbrace{x'_{4,2n-1}\, x'_{4,2n-1} \dots x'_{4,1}\, x'_{4,0}}_{2n\ bits} + \underbrace{\bar{x}'_{3,2n-1}\, \bar{x}'_{3,2n-2} \dots \bar{x}'_{3,1}\, \bar{x}'_{3,0}}_{2n\ bits} + 1 \right|_{2^{2n}} \\ &= \underbrace{x'_{4,2n-1}\, x'_{4,2n-1} \dots x'_{4,1}\, x'_{4,0}}_{2n\ bits} + \underbrace{\bar{x}'_{3,2n-1}\, \bar{x}'_{3,2n-2} \dots \bar{x}'_{3,1}\, \bar{x}'_{3,0}}_{2n\ bits} + 1 \end{aligned} \tag{۳۴}$$

این نکته قابل ذکر است که معادله (۳۳) با 2n بیت از x'_4 و معکوس x'_3 با در نظر نگرفتن بیت آخر و در نظر گرفتن بیت نقلی با مقدار یک، انجام می‌شود. اکنون، محاسبه نهایی از (۳۲) را می‌توان به صورت زیر ساده کرد:

$$x'' = x'_3 + (2^{2n}+1)H = x'_3 + \acute{H} \tag{۳۵}$$

از آنجایی که

$$\acute{H} = (2^{2n}+1) \underbrace{\mathrm{H}}_{2n\ bits} = \underbrace{H \mid H}_{4n\ bits} \tag{۳۶}$$

Algorithm 9_" the proposed DNA based with Adleman-Lipton"_ Reverse conversion for the moduli set $\{2^n+1, 2^n-1\}$

Procedure Calculate_x''_based_m1,2 (Input: $T_{x'_1}$, $T_{x'_2}$, n ; output: $T_{x''}$) {
  Copy (T_{v_1} , $T_{x'_2}$);
  CircleRightshift (T_{v_1});
  Copy (T_{v_2} , $T_{x'_1}$,);
  Separation ($T_{x'_1}$, {B_n}, T_{tmp});
  Separation (T_{tmp} , {0}, $T_{tmpcopy}$);

Algorithm 9_" the proposed DNA based with Adleman-Lipton"_ Reverse conversion for the moduli set $\{2^n+1, 2^n-1\}$

Merge ($T_{x_1'}$, T_{tmp});
If (Detect ($T_{tmpcopy}$) ==True)
CirecleRightshift (T_{v_2});
NOT_opertaion (T_{v_2});
Else
For j=0 to n-2 do
Separation (T_{v_2}, $\{B_j\}$,T^j_{tmpx1});
ValueAssignment (T^j_{tmpx1} , 1);
Merge (T_{v_2},T^j_{tmpx1});
End for;
Separation (T_{v_2}, $\{B_{n-1}\}$,T^{n-1}_{tmpx1});
ValueAssignment (T^{n-1}_{tmpx1} , 0);
Merge (T_{v_2},T^{n-1}_{tmpx1});
End if;
Add_mod2nm1 (T_{v_1},T_{v_2}, n, T_{sum});
Copy (T'_H, T_{sum});
calculate_remain_mod2nm1 (T_{sum}, n, T_H);
For i=1 to n do
Leftshift (T'_H);
End for;
Add ($T_{x_1'}$, T_H, n + 1, T_{sum1}, $T_{overflow}$);
Add (T_{sum1}, $T_{H'}$, n + 1, $T_{x''}$, $T_{overflow}$);
Discard ($T_{overflow}$) ;}

Algorithm 10_" the proposed DNA based with Adleman-Lipton"_ Reverse conversion for the moduli set $\{2^{2n}+1, 2^{2n}\}$

Procedure Calculate_x''_based_m3,4 (input: $T_{x_2'}$, $T_{x_4'}$, n ; output: $T_{x''}$) {
For j=0 to 2n-1 do
Separation$T_{x_3'}$, $\{B_i\}$, T_{tmp});
End for;
Copy ($T_{x3pcopy}$, T_{tmp});
Merge ($T_{x_3'}$, T_{tmp});
Not_operation ($T_{x3pcopy}$);
Add ($T_{x_4'}$,$T_{x3pcopy}$, 2n, T_{sum}, $T_{overflow}$);
Add (T_{sum}, 1, 2n, T_H, $T_{overflow}$);
Empty (T_{sum});
Copy (T_{Hpcopy}, T_H);
For i=1 to 2n
Leftshift (T_{Hpcopy});
End for;
Add ($T_{x_3'}$, T_{Hpcopy}, 2n, T_{sum}, $T_{overflow}$);
Add (T_{sum}, T_H, 2n,$T_{x''}$, $T_{overflow}$);
Discard ($T_{overflow}$) ;}

تابع تشخیص و تصحیح خطا در مدل ادلمن – لیپتون

بر اساس توابع گفته شده در قسمت‌های قبلی و الگوریتم تشخیص و تصحیح خطا ، این الگوریتم با DNA، دو خطا را تشخیص و یک خطا را تصحیح می‌کند. ورودی‌های این تابع شامل اعداد مانده‌ای $T_{in}^{1}, \dots, T_{in}^{count}$ و چهار پیمانه $T_{m_1}, T_{m_2}, T_{m_3}, T_{m_4}$ می‌باشد. ابتدا تابع جمع چند عدد مانده‌ای اجرا می‌شود؛ سپس بر روی حاصل عملیات این تابع عملیات تشخیص و تصحیح انجام می‌شود. الگوریتم این تابع با DNA در الگوریتم ۱۱ نشان داده شده است.

Algorithm 11_" Adleman-Lipton"_ Error detection & correction

```
Procedure detection_ correction_ DNA
(input: T_x'1 , T_x'2 , T_x'3 , T_x'4 , T_m1 , T_m2 , T_m3, T_m4, n ){
 Calculate_x''_CRT1_m1&m2 ( T_x'1 , T_x'2 , n , T_x''1,2) ;
 Subtraction ( T_x''1,2 , T_x'm3 , Tdelta3);
Subtraction (T_x''1,2 ,T_x'm4 ,Tdelta4);
Compare_twobinarynumbers (Tdelta3, 0, n+1, Tf3, Te3);
Compare_twobinarynumbers (Tdelta4, 0, n+1, Tf4, Te4);
If (Detect (Te3) =True and Detect (Te4) =True)
   No Error;
ElseIf (Detect (Te3) = false and Detect (Te4) =false)
   Calculate_x''_CRT1_m3&m4 (T_x'3 , T_x'4 ,2 n , T_x''3,4) ;
  Calculate_remainmod2np1 (T_x''3,4, 2n, T_x''m1);
  Calculate_remainmod2nm1 (T_x''3,4, 2n, T_x''m2 );
  Subtraction ( T_x''m1, T_x'm1, n, Tdelta1);
  Subtract (T_x''m2, T_x'm2, n, Tdelta2) ;
   Compare_twobinarynumbers (Tdelta1, 0, n+1, Tf1, Te1);
  Compare twobinarynumbers (Tdelta2, 0, n+1, Tf2, Te2);
    If (Te1 not empty and Te2 not empty)
      More than one error
    ElseIf (Te1 not empty)
            Copy (T_x'1, T_x''m1 );
        ElseIf (Te2 not empty)
             Copy ( T_x'2 , T_x''m2);
           End if;
       End if;
    End if;
 Else If (Te3 not empty or Te4 not empty)
      If (Te3 not empty)
           Copy (T_x'3,T_x''1,2);
      ElseIf (Te4 not empty)
          Combine (T_x'4;T_x''1,2);
         End if;
     End if;
   End if;
 End if;
Endif ;}
```

۵- ارزیابی کار پیشنهادی

در مقالات [۹] و [۱۰] محاسبات DNA با استفاده از سیستم اعداد مانده‌ای انجام شده است که کار پیشنهادی با این دو مقاله در جدول زیر مقایسه شده است.

جدول ۱: مقایسه کار پیشنهادی

مقاله	سیستم تشخیص خطا	تعداد تشخیص خطا	سیستم تصحیح خطا	تعداد تصحیح خطا	قابلیت ها بر اساس نوع پیمانه
[۱۰]	✗	-----	✗	-----	محدوده دینامیکی پایین موازی سازی پایین سرعت پایین
[۹]	✓	یک خطا	✗	-----	محدوده دینامیکی پایین موازی سازی پایین سرعت پایین
کار پیشنهادی	✓	دو خطا	✓	یک خطا	محدوده دینامیکی بالا موازی سازی بالا سرعت بالا پیاده سازی بهتر در DNA

همان‌طور که در جدول بیان شده است، سیستم کار پیشنهادی هم قابلیت تشخیص دو خطا و هم تصحیح یک خطا را دارد، که در کارهای قبلی هیچ کدام از دو قابلیت تشخیص و تصحیح را ندارد یا فقط قابلیت تشخیص را دارد و همچنین با توجه به نوع پیمانه‌ای که در کار پیشنهادی استفاده شده است، باعث کارآمدی بیشتر این سیستم می‌شود. در واقع انتخاب مناسب یک مجموعه پیمانه نقش مهمی را در طراحی سیستم‌های سیستم اعداد مانده‌ای ایفا می‌کند؛ زیرا سرعت واحد ریاضی سیستم اعداد مانده‌ای و نیز پیچیدگی مبدل باقی‌مانده به باینری بستگی به شکل و تعداد پیمانه‌ها دارد. یکی دیگر از موارد مهم در طراحی مبدل معکوس، انتخاب یک الگوریتم تبدیل مناسب است. بنابراین مجموعه پیمانه چهارتایی $\{2^n+1, 2^n-1, 2^{2n}+1, 2^{2n}\}$ را انتخاب کرده؛ چون محدوده دینامیکی آن بالا است و همچنین فرمول‌های مبدل معکوس ساده‌تری دارد که این فرمول‌ها مبدل معکوس قابلیت پیاده‌سازی بهتری در DNA دارند. این مجموعه پیمانه 6n بیتی شامل پیمانه‌های ساده و خوش‌فرم هست که دارای مبدل معکوس موثری می‌باشد و همچنین یک مجموعه پیمانه conversion friendly هست که دارای خصوصیات سرعت بالا و طراحی

مبدل معکوس با هزینه کم می‌باشد. در این مجموعه پیمانه از مبدل معکوس CRT1 استفاده می‌شود که باعث عملکرد بالا می‌شود. این مبدل تاخیر تبدیل کم و نیازمندی سخت افزاری کمتری را دارد.

در مقالات [۸،۱۰،۲۱،۲۲] الگوریتم‌هایی جهت عملیات منطقی و حسابی در محاسبات DNA تعریف شده است که در بخش ۲-۳ شرح داده شده است. در کار پیشنهادی الگوریتم‌های جدید تعریف شده است که جهت پیاده سازی عملیات جمع پیمانه‌ای به کار می‌روند.

۶- نتیجه‌گیری

در این کار، مدل Adleman-Lipton برای اجرای عملیات DNA و سیستم تشخیص و تصحیح خطا استفاده شده است. عملیات محاسباتی موردنیاز برای تشخیص و تصحیح خطاها شامل عملیات جمع دو عدد RNS است. برای تشخیص دو خطا و تصحیح یک خطا، از RNS با مجموعه ۴ پیمانه و محدوده دینامیکی 6n بیت استفاده شده است. دو پیمانه اول این مجموعه پیمانه‌های اطلاعاتی هستند و دو پیمانه دیگر برای تشخیص دو خطا و تصحیح یک خطا پیمانه افزونه هستند. مجموعه پیمانه استفاده شده در اینجا $\{2^n-1, 2^n+1, 2^{2n}+1, 2^{2n}\}$ با قابلیت اجرای موازی بالا و از نظر سرعت و هزینه کارآمد است. مبدل معکوس استفاده شده در این RNS از نوع New CRT1 است. برای انجام عملکرد تشخیص و تصحیح، الگوریتم‌های جدیدی در DNA تعریف شده است.

مراجع

[1] H. M. H. Babu. "Multiple-Valued Computing in Quantum Molecular Biology: Arithmetic and Combinational Circuits". CRC Press, 2023.

[2] S. Minocha, S. Namasudra. "Research challenges and future work directions in DNA computing." Advances in Computers 129 (2023): 363-387.

[3] C. Zhang, G. Lulu, Z. Yuchen, S. Ziyuan,Z. Zhiwei, Zaichen, Y. Xiaohu You. "DNA computing for combinational logic." Science China Information Sciences 62 (2019): 1-16.

[4] F. Hochella, M., "There's plenty of room at the bottom: Nanoscience in geochemistry," Elsevier, 2001.

[5] L. M. Adleman. "Molecular computation of solutions to combinatorial problems. Science", 266(5187), 1021-1024, 1994.

[6] L. Sousa. "Nonconventional computer arithmetic circuits, systems and applications." *IEEE Circuits and Systems Magazine* 21, no. 1 (2021): 6-40.

[7] CH. Chang, A. Sabbagh Molahosseini, A. Emrani Zarandi, TF. Tay, "Residue Number Systems: A New Paradigm to Data path Optimization for Low-Power and High-Performance Digital Signal Processing Applications," IEEE Circuits and systems magazine, vol. 15, no. 4, pp. 26-44, 2015.

[8] B. Yurke, A. J. Turberfield, A.P. Mills Jr, F. C. Simmel, J. L. Neumann, "A DNA-fuelled molecular machine made of DNA," Nature, vol. 406, no. 6796, pp. 605-608, 2000.
[9] X. Zheng, B.Wang, C. Zhou, X. Wei, Q. Zhang, "Parallel DNA Arithmetic Operation With One Error Detection Based on 3-Moduli Set," IEEE transactions, vol. 15, no. 5, pp. 499-507, 14 June 2016
[10] X. Zheng, J. Xu, W. Li, "Parallel DNA arithmetic operation based on the n-moduli set," Applied Mathematics and Computation, vol. 212, no. 1, Jun 2009, pp. 177-184.
[11] F. Famoori, A. Sabbagh Molahosseini. A, and A. Alsadat Emrani Zarandi. "DNA Arithmetic With Error Correction." *IEEE Transactions on NanoBioscience* 22.2 (2022): 329-336.
[12] M. Sarkar, P Ghosal, SP Mohanty, "Exploring the Feasibility of a DNA Computer: Design of an ALU using Sticker Based DNA Model," IEEE Transactions on Nanobioscience, vol. 16, no. 20, pp. 383-399, 2017.
[13] A. Fujiwara, KI. Matsumoto, W. Chen, "Procedures for logic and arithmetic operations with DNA molecules," International Journal of Foundations of Computer Science, vol. 15, no. 3, 2004.
[14] A. Molahosseini, L. Sousa, C. Chang (Eds), "Embedded Systems Design with Special Arithmetic and Number Systems", Springer, 2017.
[15] L Sousa, S Antao, P Martins, "Combining Residue Arithmetic to Design Efficient Cryptographic Circuits and Systems", IEEE Circuits and Systems Magazine, 16 (4), pp. 6-32, 2016.
[16] F. J. Taylor, "Residue Arithmetic A Tutorial with Examples," IEEE Computer, vol. 17, no. 5, pp. 50–62, 1984.
[17] EB Olsen, "Introduction of the Residue Number Arithmetic Logic Unit with Brief Computational Complexity Analysis (Rez-9soft processor)," Whitepaper, Digital System Research 2015.
[18] Ds. Anderson, "Design and Implementation of an Instruction Set Architecture and an Instruction Execution Unit for the REZ9 Coprocessor System," M.S. Thesis, U of Nevada LV, 2014.
[19] K. Navi, A. Sabbagh Molahosseini. M. Esmaeildoust, "How to Teach Residue Number System to Computer Scientists and Engineers," IEEE Transactions on Education, vol. 54, no. 1, pp. 156-163, 2011.
[20] T. F. Tay and C. H. Chang, "A non-iterative Multiple Residue Digit Error Detection and Correction Algorithm in RRNS," IEEE Transactions on Computers, vol. 65, no. 2, pp. 396-408, February 2016
[21] F. Guarnieri, M. F. Liss, C. Bancroft, "Making DNA add," Science, vol. 273, no. 5272, pp. 220-223, 1996.
[22] A. Fujiwara, S. Kamio," Procedures for Multiple Input Functions with DNA strands," Journal of Foundations of Computer Science 2005.
[23] W. Wang, M. N. S. Swamy, and M. Omair Ahmad. "Moduli selection in RNS for efficient VLSI implementation." 2003 IEEE International Symposium on Circuits and Systems (ISCAS). Vol. 4. IEEE, 2003.
[24] A. Sabbagh Molahosseini, K.Navi, C. Dadkhah, O. Kavehei, S. Trimarchi, "Efficient Reverse Converter Designs for the New 4-Moduli Sets {2n–1, 2^n, 2^n+1, 2^{2n+1}–1} and {2^n–1, 2^n+1, 2^{2n}, 2^{2n}+1} Based on New CRTs," IEEE Transactions on Circuits and Systems-I, vol. **57**, no. 4, pp. 823-835, 2010.

[25] B. Deng, S. Srikanth, E. R. Hein, T. M. Conte, E. Debenedictis, Je. Cook, M.P. Frank, "Extending Moore's Law via Computationally Error Tolerant Computing," ACM Transactions on Architecture and Code Optimization (TACO), vol. 15, no. 1, 2018.