علوم و مهندسی کامپیوتر
نشریه علمی پژوهشی انجمن کامپیوتر ایران
مجلد ۹، شماره ۱ و ۳ (الف)، ۱۳۹۰
صفحات ۶۵-۷۹
مقاله عادی

افزایش قابلیت اطمینان ریزپردازنده با رویکرد ریزمعماری تصحیح اشکالات در دستورها

شیلان پارساییان[۱] امیر رجب‌زاده[۲]

[۱]دانشکده فنی و مهندسی، دانشگاه چالمرز، سوئد
[۲]دانشکده فنی و مهندسی، دانشگاه رازی، کرمانشاه، ایران

چکیده

گسترش کاربرد سیستم‌های کامپیوتری تعبیه شده (Embedded) در حوزه‌های مختلف زندگی بشر باعث شده است که بروز اشکال (Fault) و در نتیجهٔ آن خطا (Error) و ازکارافتادگی (Failure) ، در اغلب این سیستم‌ها، صدمات جبران ناپذیری را به دنبال داشته باشد. این مقاله، روشی با نام PFC (Pipeline Fault Correction) در سطح معماری برای تصحیح اشکالات چندبیتی با استفاده از کد همینگ در خط لولهٔ ریزپردازندهٔ تعبیه شده ارائه می‌دهد. این روش دارای دو مکانیزم است.
مکانیزم اول، BMBC (Branch Multi-Bit Correction)، محافظ اشکالات چندبیتی در دستورهای انشعاب بوده و قبل از این دستورها، یک دستور خاص درج می‌کند و قادر به تصحیح حداکثر ۸ بیت اشکال همزمان در هر دستور انشعاب است. مکانیزم دوم، OSBC (Opcode Single-Bit Correction)، محافظ اشکالات یک‌بیتی در کد عملیاتی کلیه دستورهای برنامه است. این مکانیزم برای تصحیح یک بیت اشکال در کد عملیاتی، یک دستور خاص دیگر را هر شش خط یک بار در برنامه درج می‌کند. در این مقاله برای مقایسه، روش دیگری به نام MFC (Memory Fault Correction) برای تصحیح اشکالات چندبیتی با استفاده از کد همینگ و صرفاً در حافظهٔ یک سیستم تعبیه شده، پیاده‌سازی شده است. در این روش، کدهای توازن به انتهای هر کلمه از حافظه افزوده شده، بطوریکه به صورت مشابه، قادر به تصحیح ۸ اشکال همزمان در هر دستور یا داده از حافظه باشند.
هر دو روش، PFC و MFC، به طور مستقل بر روی ریزپردازندهٔ OpenRISC طراحی و سنتز شده‌اند. و نتایج ارزیابی این دو روش استخراج و مقایسه شده‌اند. نتایج ارزیابی تحلیلی نشان می‌دهد که پوشش تصحیح اشکال مکانیزم BMBC برای یک تا هشت اشکال همزمان در یک دستور انشعاب بین ۱۰۰٪ تا ۲/۳۱٪ و در مکانیزم OSBC برای یک اشکال در کد عملیاتی هر دستور ۱۰۰٪ است. در روش MFC پوشش تصحیح اشکال برای یک تا هشت اشکال همزمان در یک کلمه از حافظهٔ دستور یا داده بین ۱۰۰٪ تا ۲/۳۱٪ است. نتایج حاصل از سنتز ریزپردازنده بر روی تکنولوژی TSMC 0.18um، حاکی از آن است که در روش PFC، سخت‌افزار به کار رفته به میزان ۱۱/۱۰٪ و توان مصرفی به میزان ۳/۸۱٪ نسبت به ریزپردازندهٔ اولیه، افزایش یافته است، این در حالی است که در روش MFC، سربار سخت‌افزار و میزان توان مصرفی به ترتیب ۱۳/۱۴٪ و ۱۴/۲۶٪ افزایش پیدا کرده‌است. همچنین نتایج ارزیابی تجربی نشان می‌دهد که در روش PFC سربار حجم حافظه و سربار زمان اجرا نسبت به ریزپردازندهٔ اولیه به ترتیب ۳۱/۶۷٪ و ۳۵/۹۸٪ است. در روش MFC سربار حجم حافظه ۷۵٪ است.

کلمات کلیدی: کشف و تصحیح اشکال، خطای روند اجرای برنامه، ریزپردازندهٔ تحمل پذیر اشکال، بازیابی سیستم، کد همینگ.

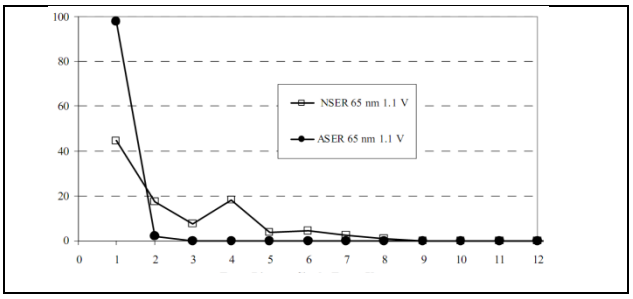
۱- مقدمه

افزایش نیاز محاسباتی و تمایل به ریزپردازنده‌هایی با کارایی بیشتر و همچنین گسترش همه جانبهٔ حوزه‌های کاری سیستم‌های مبتنی بر ریزپردازنده، پیشرفت مداوم در صنعت میکروالکترونیک را در پی دارد. طول کانال ترانزیستورها و ولتاژ تغذیهٔ آن‌ها به طور مداوم در حال کاهش و فرکانس کاری مدارهای تراشه‌ها در حال افزایش است. این تغییرات، حساسیت تراشه‌ها را نسبت به اشکالات[۱] ناشی از عوامل محیطی از قبیل یون‌های سنگین، تشعشات الکترومغناطیسی و اغتشاشات منبع تغذیه افزایش داده است [۱]، [۲].

از طرفی امروزه دامنهٔ کاربرد سیستم‌های تعبیه شدهٔ[۲] مبتنی بر ریزپردازنده‌ها، گسترش فراوانی پیدا کرده است، بطوریکه بیش از ۷۹٪ از ریزپردازنده‌های تولید شده، در سیستم‌های تعبیه شده بکار می‌روند [۳]. در این کاربردها، اشکالات رخدادهٔ ناشی از عوامل محیطی و ازکارافتادگی سیستم، می‌تواند خطرات جانی، ضررهای مالی و از بین رفتن اطلاعات را در پی داشته باشد. لذا امروزه، طراحی یا بکارگیری مکانیزم‌های کشف اشکال و بازیابی سیستم پس از رخداد اشکال، به ارکان طراحی سیستم‌های تعبیه شده تبدیل گردیده است.

اشکالات رخ داده در یک سیستم تعبیه شده، ممکن است در حافظهٔ دستور، حافظهٔ داده و یا مدارهای داخلی آن بوجود آیند. اشکالات ناشی از عوامل محیطی، در سطح سیلیکن تغییرات گذرایی را بوجود می‌آورند و از آنجاییکه تنوع این اشکالات در سطح سیلیکن بسیار زیاد است، همچنین معمولاً مجموعه دقیقی از آن قابل تعریف نیست، لذا مدل‌سازی این اشکالات در سطوح تجرید بالاتر، تنوع آن را کاهش داده و امکان تعریف مجموعهٔ دقیقی از آن را میسر می‌کند. یکی از مشهورترین مدل‌های اشکال در سطح گیت و معماری، اشکال SEU[۳] است [۴]. اشکالات SEU، می‌توانند به صورت مستقیم، مقدار بیت دادهٔ یک خانهٔ حافظه و یا یک ثبات را تغییر دهند، یا در مدارهای ترکیبی، یک پالس گذرا تولید کنند، که این تغییر ناخواسته می‌تواند نهایتاً در یک المان حافظه‌ای ذخیره گردد [۴].

اشکالات SEU، ممکن است به صورت اشکال واژگونی یک‌بیتی[۴] (SBF) یا چندبیتی[۵] (MBU) حادث گردد. در گذشته اشکالات SBF از اهمیت بالائی برخوردار بودند ولی امروزه اشکالات MBU، بسیار مهم بوده و ره‌آورد جدیدی از پیشرفت صنعت میکروالکترونیک است. در این اشکال، تعداد بیت‌های صدمه دیده، از یک بیت بیشتر است. این اشکالات امروزه به دلیل افزایش فشردگی مدارها مورد توجه قرار گرفته‌اند [۴]. با توجه به تحقیقات کنونی [۵] و نتایج حاصل شده از پرتاب ذرات نوترون (NSER) و آلفا (ASER) بر روی حافظه‌های SRAM با تکنولوژی ۶۵ نانومتر، همانطور که در شکل ۱ آمده است، تنها ۴۵٪ از اشکالات رخ داده در اثر تابش ذرات نوترون، SBF هستند. در این شکل، محور عمودی درصد مشاهدهٔ اشکال و محور افقی تعداد بیت‌های اشکال را نمایش می‌دهد. اگرچه اشکالات MBU تا ۲۰ بیت اشکال همزمان هم مشاهده شده است ولی اشکالات MBU بزرگتر از ۸ بیت اشکال همزمان در معرض ذرات نوترون، و بزرگتر از ۲ بیت اشکال همزمان در معرض ذرات آلفا، بسیار ناچیز است.



شکل ۱- اشکالات SBF و MBU ناشی از تابش ذرات سنگین بر روی SRAM

مدل اشکال در این مقاله، اشکال SEU در مدارهای داخلی و در حافظه است که به صورت اشکال واژگونی یک‌بیتی (SBF) و چندبیتی (MBU) در نظر گرفته شده است. یکی از نکات برجسته روش پیشنهادی در این مقاله، قابلیت مقابله با اشکالات MBU است که امروزه از اهمیت خاصی برخوردار شده‌اند.

یکی از روش‌های مقابله با اشکالات ناشی از عوامل محیطی روش کشف همروند خطا در سیستم‌های کامپیوتری است که می‌توان آن را به دو دسته تقسیم کرد [۶]:

۱) روش‌های سطح مدار[۶]: از روش‌های سطح مدار می‌توان به استفاده از کدهای تصحیح خطا در حافظه‌ها، بیت‌های توازن در گذرگاه‌های داده، کدهای residue در ALU و مدارهای self-checking اشاره نمود.

۲) روش‌های سطح سیستم[۷]: از روش‌های این سطح می‌توان به زمان‌سنج مراقب، ریزپردازندهٔ مراقب، ساختمان دادهٔ تحمل‌پذیر خطا، تکرار (مثل FTMP و SIFT و همچنین N-version programming) اشاره نمود.

همانطور که اشاره شد، استفاده از کدهای تصحیح خطا یکی از روش‌های کشف همروند خطا در سطح مدار است. کدهای تصحیح خطا می‌توانند در سطح مدارهای داخلی ریزپردازنده[۸] یا در سطح مدارهای خارجی ریزپردازنده[۹] و بر روی حافظه پیاده‌سازی شوند. ریزپردازنده‌های طراحی شده به این روش می‌توانند به صورت تراشه‌های ASIC تولید شوند، مانند LEON [۷]، یا به صورت قسمتی از IP-Core طراحی شده و در اختیار استفاده‌کنندگان قرار گیرند [۸].

در این مقاله، با استفاده از کد همینگ یک روش در سطح مدارهای داخلی تراشه برای تصحیح خطای یک ریزپردازندهٔ تحمل‌پذیر اشکال مبتنی بر IP-Core ارائه شده است.

این روش PFC (Processor Pipeline Fault Correction) نام دارد، سیستم کشف و تصحیح اشکال در خط لولهٔ ریزپردازنده پیاده‌سازی شده است. PFC دارای دو مکانیزم است مکانیزم اول و قسمتی از مکانیزم دوم برای تصحیح اشکالاتی است که منجر به خطای روند اجرا می‌شوند. مکانیزم اول، BMBC[۱۰]، یک دستور خاص قبل از هر دستور انشعاب و مکانیزم دوم، OSBC[۱۱]، یک دستور خاص دیگر را هر شش خط یک بار در برنامه درج می‌کند. سخت‌افزار طراحی شدهٔ داخلی ریزپردازنده، ۸ اشکال همزمان در هر دستور انشعاب و یک اشکال در کد عملیاتی هر دستور در برنامه را تصحیح می‌کند. در این مقاله همچنین روش دیگری به نام MFC (Memory Fault Correction)، برای تصحیح اشکالات چندبیتی در حافظهٔ سیستم با استفاده از کد همینگ، به منظور مقایسه با روش PFC پیاده‌سازی شده است. MFC در سیکل نوشتن، کد تصحیح کنندهٔ هر کلمه از حافظه را به انتهای آن می‌افزاید و در حافظه ذخیره می‌کند. به این ترتیب در سیکل خواندن، این کد می‌تواند در هر دستور از حافظهٔ دستور[۱۲] یا هر داده از حافظهٔ داده[۱۳]، ۸ اشکال همزمان را کشف و تصحیح کند.

هر یک از این روش‌ها، PFC و MFC، به طور مستقل با زبان توصیف سخت‌افزار Verilog بر روی ریزپردازنده OpenRISC و حافظهٔ آن [۹] پیاده‌سازی شده‌اند و صحت عملکرد آن‌ها تحقیق شده و نتایج حاصل از آن‌ها مقایسه شده است. نتایج ارزیابی تحلیلی نشان می‌دهند که پوشش تصحیح اشکال نسبت به مدل در نظر گرفته شده، در مکانیزم BMBC، برای یک تا هشت اشکال همزمان در یک دستور انشعاب بین ۱۰۰٪ تا ۲/۳۱٪ و در مکانیزم OSBC برای یک اشکال در کد عملیاتی هر دستور ۱۰۰٪ است. در روش MFC نیز پوشش تصحیح اشکال برای یک تا هشت اشکال همزمان در یک کلمه از حافظهٔ دستور یا داده بین ۱۰۰٪ تا ۲/۳۱٪ است. در این روش تا ۲ بیت اشکال در کد عملیاتی دستورها با احتمال ۶۶/۶۷٪ تصحیح می‌شود و نیز برای یک تا هفت اشکال همزمان در عملوند دستورهای غیرانشعابی پوشش تصحیح اشکال بین ۱۰۰٪ تا ۴/۸۲٪ است. نتایج حاصل از سنتز ریزپردازنده توسط نرم‌افزار Synopsys Design Compiler بر روی تکنولوژی TSMC 0.18um، حاکی از آن است که در روش PFC، سخت‌افزار به کار رفته به میزان ۱۱/۱۰٪ و نیز توان مصرفی به میزان ۳/۸۱٪ نسبت به ریزپردازندهٔ اولیه، افزایش یافته است.

در روش MFC، حجم سخت‌افزار و میزان توان مصرفی به ترتیب به میزان ۱۳/۱۴٪ و ۱۴/۲۶٪ نسبت به ریزپردازندهٔ اولیه افزایش پیدا کرده است. همچنین نتایج ارزیابی تجربی نشان می‌دهند که در روش PFC سربار حجم حافظه و سربار

زمان اجرا نسبت به ریزپردازندهٔ اولیه به ترتیب ۳۱/۶۷٪ و ۳۵/۹۸٪ است. در روش MFC سربار حجم حافظه ۷۵٪ است.

در ادامه مقاله ابتدا مروری بر کارهای پیشین خواهیم داشت. در فصل سوم مدل اشکال مورد نظر در این مقاله و در فصل چهارم و پنجم روش‌های PFC و MFC ارائه شده‌اند. در فصل ۶ نتایج حاصل از ارزیابی و در فصل ۷ بحث و مقایسهٔ روش‌ها و در نهایت در فصل ۸ خلاصه و نتیجه‌گیری بیان خواهد شد.

۲- کارهای پیشین

در این بخش مروری خواهیم داشت بر کد تصحیح خطای همینگ و سپس در ادامه روش‌های پیشین نظارت بر روند اجرای برنامه ارائه خواهد شد.

۲-۱- کد همینگ

کد همینگ یک کد باینری کشف و تصحیح خطاست که در آن رابطهٔ $d + p + 1 \leq 2^p$ برقرار است. در این رابطه d تعداد بیت‌های داده است و p تعداد بیت‌های parity است. با برقراری این رابطه، کد همینگ قادر است تمام خطاهای یک بیتی را در یک کلمهٔ d+p بیتی تصحیح کند و یا تمام خطاهای دو بیتی را در این کلمه کشف کند (SEC-DEC) [۱]. ساختار این کد مجموعه‌ای از داده‌ها و بیت‌های توازن است که بیت‌های توازن در مرحلهٔ کدگذاری از داده‌های اصلی بدست می‌آیند و به همراه دادهٔ اصلی ارسال می‌شوند. سپس در مرحلهٔ تصحیح اشکال، مجدداً این بیت‌های توازن از داده استخراج می‌شوند و با بیت‌های اخذ شدهٔ قبلی که به همراه داده ارسال شده‌اند، مقایسه می‌شوند. به عبارتی در این مرحله برای تشخیص و تصحیح خطا، بیت‌های سندروم (S) از XOR بیت‌های توازن دریافت شده و بیت‌های توازن تولید شده در مقصد حاصل می‌شود. تمام بیت‌های سندروم برای حالتی که خطایی رخ نداده باشد صفر هستند. برای رخداد یک بیت خطا، سندروم غیر صفر بوده و شمارهٔ مکان بیت خطا را نشان می‌دهد. برای دو بیت خطا مقدار سندروم غیر صفر است اما مقدار آن نشان دهندهٔ شمارهٔ محل بیت خطا نمی‌باشد.

۲-۲- روش‌های کشف و تصحیح همروند خطا

روش‌های رفتاری کشف خطا مشهورترین روش‌های کشف همروند خطای[۱۴] در سطح سیستم هستند [۶]. در این روش‌ها رفتاری از سیستم مد نظر قرار می‌گیرد. این رفتار طبق الگویی در زمان کامپایل استخراج و در زمان اجرا مجدداً استخراج و با رفتار استخراجی از زمان کامپایل مقایسه می‌گردد. این رفتارها می‌توانند زمان اجرای قطعه‌ای از برنامه [۱۰]، ترتیب اجرای دستورهای برنامه [۱۱]، دسترسی به حافظه [۱۲]، نظارت بر امضاهای از قبل محاسبه شده [۱۳] و ... باشند. در این میان روش کنترل روند اجرا که بر اساس روش نظارت بر امضا است بسیار کارآمد است. بیش از ۷۰٪ از خطاهای موثر در ریزپردازنده به خطاهای روند اجرای برنامه منجر می‌گردند [۱۴] و کشف آنها بیش از ۷۰٪ کل خطاهای موثر را پوشش می‌دهد. سیستم‌های تحمل پذیر خطا که از روش‌های کشف خطای CFE بهره می‌گیرند را می‌توان به سه دسته تقسیم نمود [۱۵]: ۱) روش طراحی با ریزپردازنده خاص منظوره، ۲) روش طراحی با قابلیت‌های خاص ریزپردازنده عام منظوره و ۳) روش طراحی با قابلیت‌های عمومی ریزپردازنده عام منظوره.

طراحی با ریزپردازنده خاص منظوره[۱۵]: در این روش از ریزپردازندهٔ خاص تحمل‌پذیر خطا استفاده می‌شود و یا یک ریزپردازندهٔ خاص تحمل‌پذیر خطا طراحی می‌شود. این ریزپردازنده‌ها می‌توانند به صورت ASIC تولید شوند یا به صورت قسمتی از IP-Core در اختیار استفاده‌کنندگان قرار گیرند. در روش‌های مبتنی بر IP-core که مد نظر این مقاله نیز هست با کمک نرم‌افزارهای شبیه سازی (Modelsim، Icarus، Quartus و ...) و زبان‌های توصیف سخت‌افزاری چون Verilog، VHDL ریزپردازندهٔ جدیدی با قابلیت تحمل‌پذیری خطا تولید می‌شود. کد تعدادی از ریزپردازنده‌ها به صورت منبع باز در اختیار کاربران قرار گرفته است [۱۵] که با تغییر دادن این کد می‌توان به اهداف مورد نظر دست یافت. این روش‌ها از نظر پوشش کشف خطا و کاهش سربار زمان اجرا و سربار حافظه در وضعیت بهینه‌ای قرار دارند [۱۶]، [۱۷] ، [۱۸]. عدم پوشش خطای MBU در اغلب روش‌های [۱۶] کنونی و نیز عدم کشف خطای درج پرش در آن‌ها (تبدیل دستور غیرانشعابی به دستور انشعابی با رخداد خطا در کد عملیاتی دستور غیرانشعابی) [۱۶]، [۱۷]، [۱۸] از مشکلاتی هستند که در این مقاله بررسی شده و حل شده‌اند. در ادامه چند روش مشهور از این دسته به طور مختصر توضیح شده است.

روش Roll-back Based Recovery Procedure [۱۹] با دو خط لوله در ریزپردازنده به کشف و تصحیح خطای دستور می‌پردازد. در واقع در این روش ریزپردازنده دو شمارندهٔ دستور دارد که هر یک متعلق به یک خط لوله است. با فراخوانی هر دستور که شمارنده‌ها آدرس آن را تولید کرده‌اند دستور وارد خط لوله می‌شود و سپس در هر طبقه از خط لوله‌ها دستورهای طبقه‌های معادل با هم مقایسه می‌شوند. عدم برابری دستورهای طبقات معادل، باعث برگشت دور ریختن دستور فعلی و بازگشت آخرین دستور صحیح در آن طبقه خواهد شد. این روش به دلیل استفاده از مکانیزم Roll-back سربار زمان اجرای بالایی دارد.

طراحی پردازندهٔ تحمل پذیر خطای LEON-FT [۲۰]، روشی مبتنی بر IP-core است که برای کشف و تصحیح خطای داده در ریزپردازندهٔ ASIC، LEON طراحی شده است. LEON ریزپردازنده‌ای خاص منظوره است که اغلب در کاربردهای فضایی استفاده می‌شود. در ریزپردازندهٔ LEON اولیه، از نیمه هادی مقاوم به SEU استفاده شده است که این ساختار تا حدودی مانع از نفوذ و گسترش خطا در نیمه هادی خواهد بود. اما صرفاً استفاده از نیمه هادی مقاوم، پوشش کشف خطای بالایی ارائه نمی‌دهد. قصد اصلی طراح در این روش، حفاظت داده در برابر نفوذ SEU بوده است. بنابراین برای این منظور از کدهای تصحیح خطایی چون parity (با ۲ بیت، زوج و فرد) در حافظهٔ نهان، روشی به نام BCH با ۷ بیت برای حفاظت بانک ثبات و روش سه‌تائی TMR[۱۶] برای محافظت ثباتهای دیگر استفاده شده است. این روش‌ها در ریزپردازندهٔ LEON، هم خطای SBF و هم خطای MBU را پوشش داده‌اند. در این پردازنده، فرآیند تشخیص خطا به بخش اجرا منتقل شده است و به موازات اجرای دستور، صحت داده هم بررسی می‌شود. به محض شناسایی یک خطا در بخش اجرا، کل خط لوله تخلیه می‌شود که البته این نحوهٔ پیاده‌سازی هم به نوبهٔ خود موجب اتلاف زمان و توان خواهد بود. بعد از این مرحله، با توجه به استفاده از کدهای تصحیح خطا، دادهٔ صحیح در ثبات ذخیره می‌شود و خط لوله از جایی که دستور با داده غلط اجرا شده است دوباره شروع به واکشی دستور می‌کند و این بار دستور احتمالاً با دادهٔ صحیح اجرا خواهد شد.

پردازنده‌های ERC32 و Thor [۲۱] از دستهٔ ریزپردازنده‌های تحمل‌پذیر خطا هستند. در این ریزپردازنده‌ها از بررسی امضا برای کشف خطا استفاده شده است. در واقع در این ریزپردازنده‌ها از سخت‌افزار تولید کد چرخشی[۱۷] استفاده شده است. این سخت‌افزار در زمان اجرای برنامه، از کد عملیاتی دستورهای اجرا شده، تمام محتوای دستورهای اجرا شده و یا آدرس دستورهای اجرا شده یک امضای فشرده شده، CRC [۲۱]، تولید می‌کند. در این روش نقاط خاصی از برنامه به عنوان نقاط وارسی[۱۸] در نظر گرفته می‌شوند. در این نقاط امضای تولید شده در زمان اجرای برنامه، با امضای مورد انتظار مقایسه می‌شود و در صورتی که این دو امضا تفاوت داشته باشند، وجود خطا تشخیص داده می‌شود. در صورتی که خطایی وجود

نداشته باشد، ثبات مربوط به امضا پاک می‌شود و آمادهٔ تولید امضای جدید می‌شود. در این روش تأخیر کشف خطا بالاست و تا زمان رسیدن به یک نقطهٔ وارسی بروز خطا مشخص نخواهد شد.

روش Hardware Assisted Pre-Emptive (HAP) [۱۷] از هر دستور در زمان کامپایل دو نسخه تهیه می‌کند و درست قبل از دستور اصلی در حافظه دستور قرار می‌دهد. پرچمی[۱۹] با نام F به سخت‌افزار اضافه شده است (می‌توان از توسعهٔ پرچم‌های موجود برای آن استفاده کرد) که برای تشخیص نسخهٔ اصلی یا فرعی بودن دستورها استفاده می‌شود. اگر این پرچم ۱ باشد به معنی اصلی بودن دستور است و اگر صفر باشد به معنی فرعی بودن دستور خواهد بود. دستورها در مرحله اجرا با هم مقایسه می‌شوند. نتیجه مقایسه دستور اصلی و دستور فرعی می‌تواند یا در جهت کشف خطا و توقف فرآیند استفاده شود و یا برای اصلاح دستور اصلی به فرض صحیح بودن دستور فرعی باشد.

روش IMCFEDC [۸] قبل از هر دستور انشعاب یک دستور جدید که حاوی کد همینگ دستور انشعاب است قرار می‌دهد. به این ترتیب قبل از ورود دستور انشعاب به خط لوله، کد همینگ آن وارد خط لوله می‌شود. بنابراین در سیکل بعد اشکال یک بیتی احتمالی در دستور انشعاب با استفاده از کد همینگ تصحیح خواهد شد. این روش اشکالات چند بیتی را کشف نمی‌کند و نیز قادر به تصحیح اشکال درج پرش نیز نمی‌باشد.

طراحی با قابلیت‌های خاص ریزپردازنده عام منظوره: طراحی این سیستم‌ها براساس قابلیت‌ها و دستورهای خاص (مثل قابلیت Performance Monitoring) از ریزپردازنده‌های عام منظورهٔ تجاری است. شرکت‌های سازنده در برخی ریزپردازنده‌ها قابلیت‌هایی جهت آزمون کارایی[۲۰] یا عیب‌یابی[۲۱] تعبیه کرده‌اند. این قابلیت‌ها در اکثر ریزپردازنده‌های مشهور کنونی مانند AMD، Alpha، Pentium، MIPS_R1000 و POWERPC_604 قرار داده شده‌اند. به عنوان مثال قابلیت استثنای ردیابی انشعاب[۲۲] [۱۱]، شمارنده‌های نظارت بر کارایی[۲۳] [۲۲]، ردیابی اجرا[۲۴] [۱۵] را می‌توان نام برد. این روش‌ها می‌توانند سخت‌افزاری [۱۵] یا نرم‌افزاری [۱۱] یا ترکیبی از این دو [۲۲] باشند. این روش‌ها اساساً برای اهداف کنترل روند اجرا طراحی نشده‌اند، لذا ممکن است دارای پوشش کشف خطای بالایی نباشند [۱۶].

طراحی با قابلیت‌های عمومی ریزپردازنده عام منظوره[۲۵]: طراحی این سیستم‌ها براساس استفاده از ریزپردازنده‌های عام منظوره تجاری است. این روش‌ها معمولاً دارای پوشش کشف خطای بالایی نیستند. در این روش‌ها عموماً از قابلیت‌هایی که در اکثر ریزپردازنده‌ها وجود دارند استفاده می‌شود. این روش‌ها به سه دستهٔ سخت‌افزاری و نرم‌افزاری و ترکیبی[۲۶] تقسیم می‌شوند. در روش‌های سخت‌افزاری عموماً یک سخت‌افزار جانبی یا ریزپردازندهٔ مراقب[۲۷] در خارج از ریزپردازندهٔ اصلی برای کنترل رفتار ریزپردازندهٔ اصلی تعبیه شده است. در روش‌های نرم‌افزاری غالباً با اضافه کردن دستورهایی به لیست دستورهای کاربر (دستورهای عمومی) در حین اجرای برنامه، بر پارامترهای مجرد شده از سیستم نظارت می‌شود [۲۳]. مشهورترین روش در این دسته، روش نظارت بر امضا است [۱۳]. در روش‌های ترکیبی برای استفاده از مزایای هر دو روش، از ترکیب سخت‌افزار و نرم‌افزار استفاده شده است [۲۴].

۳- مدل‌های اشکال

در این مقاله اشکالات SEU در مدارهای داخلی و حافظه به عنوان مدل اصلی اشکالات گذرا در ریزپردازنده مد نظر قرار گرفته‌اند. این اشکالات می‌توانند به صورت اشکال واژگونی یک بیتی (SBF) و یا چند بیتی (MBU) ظاهر شوند. اشکالات SEU در یک دستور می‌توانند پنج نوع اشکال ایجاد نمایند:

۱) اشکال چند بیتی درج پرش[۲۸] (BIF): بروز اشکال MBU در کد عملیاتی، یک دستور غیرانشعابی را به یک دستور انشعابی تبدیل کرده است [۲۵].

۲) اشکال چند بیتی تغییر مقصد پرش[۲۹] (BTMF): بروز اشکال MBU در میدان آدرس یک دستور انشعابی، مقصد آن دستور انشعابی را تغییر داده است [۲۵].

۳) اشکال چند بیتی حذف پرش[۳۰] (BDF): بروز اشکال MBU در کد عملیاتی یک دستور انشعابی، آن دستور را به یک دستور غیرانشعابی تبدیل کرده است [۲۵].

۴) اشکال یک بیتی تغییر کد عملیاتی دستور غیرانشعابی[۳۱] (NBOMF): بروز اشکال SBF در کد عملیاتی، یک دستور غیرانشعابی را به یک دستور غیرانشعابی دیگر تبدیل کرده است.

۵) اشکال چند بیتی تغییر عملوند دستورهای غیرانشعابی[۳۲] (NBPMF): رخداد اشکال MBU در میدان عملوند دستور غیرانشعابی، مقدار بلافصل یا آدرس‌دهی ثبات عملوند را تغییر داده است.

رخداد اشکالات BIF، BTMF و BDF منجر به خطای روند اجرای برنامه می‌گردند، اما اشکالات NBOMF و NBPMF به طور مستقیم تأثیری در روند اجرای برنامه ندارند ولی ممکن است نتایج حاصل از آن‌ها بر بیت‌های پرچم تأثیر گذاشته و جریان برنامه را در انشعاب‌های بعدی منحرف سازد.

۴- روش پیشنهادی PFC: تصحیح اشکال در خط لولهٔ ریزپردازنده

این بخش، یک روش سطح معماری را برای طراحی یک ریزپردازندهٔ تحمل‌پذیر اشکال مبتنی بر IP-Core ارائه می‌دهد که در آن سیستم کشف و تصحیح اشکال در خط لولهٔ ریزپردازنده پیاده‌سازی شده است. هدف اصلی روش PFC، ایجاد سخت‌افزاری است که بتواند اشکالات ایجاد شده در اثر SEU در حافظهٔ دستور یا مدارات داخلی را کشف و تصحیح کند. برای این مهم، تغییرات در سخت‌افزار و همچنین نرم‌افزار مورد نیاز است. در این طرح، تغییرات سخت‌افزاری شامل اضافه کردن واحدهایی به خط لولهٔ ریزپردازنده به زبان Verilog نوشته شده‌اند، همچنین یک پس‌پردازشگر[۳۳] به زبان C++ برای تغییرات نرم‌افزاری در برنامه کاربر طراحی شده است. این تغییرات شامل درج دستورهایی خاص به برنامه کاربر بعد از تولید دستورهای اسمبلی توسط مترجم ریزپردازنده است. این طرح نیاز به دستورات خاص دارد که باید مجموعه دستورات ریزپردازنده اضافه شوند.

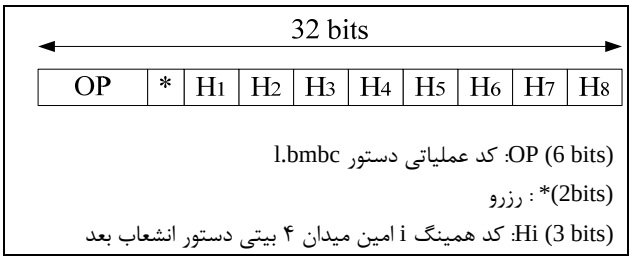
در این طرح از ریزپردازندهٔ OpenRISC استفاده شده است. این ریزپردازنده شامل دستورهایی است که می‌توانند به دلخواه کاربر تغییر کنند، در پیاده سازی این مکانیزم‌ها از این دستورها استفاده شده است. از میان این دستورها، دو دستور که کد عملیاتی آن‌ها از کد عملیاتی دستورهای معمول ریزپردازنده و نیز از یکدیگر، حداقل فاصلهٔ همینگ ۲ را دارند انتخاب شدند، به این ترتیب با بروز اشکال SBF در کد عملیاتی این دستورها، هیچ یک از دستورهای مجاز ریزپردازنده تولید نمی‌شوند و افزودن دستورهای جدید باعث افزایش حالت‌های اشکال در برنامه نخواهد شد. دستورهای یاد شده هیچ تأثیر دیگری بر عملکرد عادی ریزپردازنده نداشته و مانند دستور NOP عمل می‌کنند.

PFC شامل دو مکانیزم است که مکانیزم اول BMBC و مکانیزم دوم OSBC خوانده می‌شود. این مکانیزم‌ها قابلیت تصحیح اشکالات BIF، BTMF، BDF و NBOMF را دارند.

۴-۱- مکانیزم اول (BMBC) از روش PFC

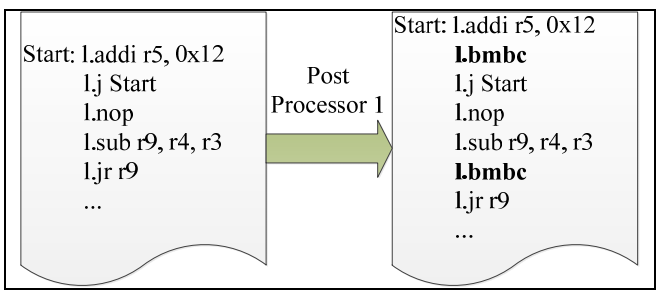
این مکانیزم جهت کشف و تصحیح اشکالات MBU در دستورهای انشعاب طراحی شده است. این مکانیزم قادر به تصحیح اشکالات نوع BTMF و BDF است.

تغییر نرم‌افزاری برای پیاده سازی مکانیزم BMBC: این مکانیزم نیاز به ایجاد تغییر در بارکاری مورد نظر دارد. پس‌پردازشگر مکانیزم BMBC یک دستور طراحی شدهٔ خاص، دستور l.bmbc، را قبل از هر دستور انشعاب در برنامه درج می‌کند. این دستور، مانند سایر دستورهای ریزپردازنده یک دستور ۳۲ بیتی است و حاوی ۸ میدان ۳ بیتی است. این سه بیت، در واقع بیت‌های توازن در ساختار کد همینگ (۴و ۷) است که در دو مرحله پردازش به برنامه کاربر افزوده شده است. ابتدا پس‌پردازشگر اول، برنامه اسمبلی حاصل از برنامه کاربر را به عنوان ورودی پذیرفته و دستورهای خام l.bmbc را قبل از هر یک از دستورهای انشعاب درج می‌کند. در مرحله بعد پس از پردازش این برنامه توسط ریزپردازنده، کد هگزادسیمال دستورهای برنامه حاصل می‌شود. در این مرحله هنوز کدهای همینگ درج نشده‌اند و چون ریزپردازنده از قبل این دستورها را می‌شناخته کد عملیاتی آن‌ها را در کد هگزادسیمال برنامه درج کرده ولی مابقی بیت‌های دستور را با مقادیر رزرو که در این ریزپردازنده "صفر" تعیین شده است پر می‌کند. در مرحلهٔ بعد با کمک پس‌پردازشگر دوم، برنامه تحلیل شده و کد همینگ دستورهای انشعاب محاسبه و طبق ساختار تعریف شده در شکل ۲ در کد هگزادسیمال دستور l.bmbc درج می‌گردد. بنابراین برنامه تمامی اطلاعات لازم جهت کشف و تصحیح اشکال را با خود خواهد داشت. دستور درج شده برای هر میدان در دستور انشعاب بیت‌های توازن کد همینگ را در بر خواهد داشت. لذا هر میدان ۴ بیتی در دستور انشعاب به طور جداگانه محافظت خواهد شد.



شکل ۲- ساختار کلی دستور l.bmbc

این دستور دارای ۸ عملوند بلافصل H_i است. هر دستور انشعاب در برنامه به میدان‌های ۴ بیتی تقسیم می‌شود و با توجه به ۳۲ بیتی بودن دستورهای ریزپردازندهٔ OpenRISC، ۸ میدان ۴ بیتی در دستور انشعاب وجود دارد. هر عملوند بلافصل H_i در دستور l.bmbc، کد همینگ (۴ و ۷)، i امین میدان ۴ بیتی دستور انشعاب بعدی است. میدان‌های H_i بیت‌های توازن ۳ بیتی کد همینگ هستند. تغییرات حاصل از پس‌پردازشگر نخست بر برنامهٔ کاربر در این مکانیزم در شکل ۳ نشان داده شده است.



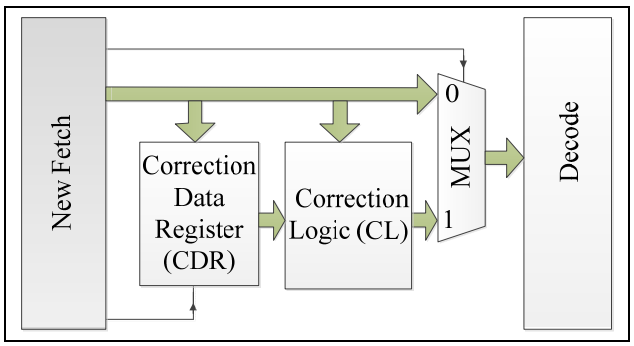
شکل ۳- تغییر در برنامهٔ بارکاری حاصل از پس‌پردازشگر اول در مکانیزم BMBC

در شکل ۳ قطعه‌ای از یک برنامهٔ بارکاری آورده شده است. مکانیزم BMBC قبل از هر دستور انشعاب (l.j و l.jr) یک دستور l.bmbc قرار می‌دهد.

قابل ذکر است که کد همینگ اضافه شده، قادر به تصحیح یک اشکال در هر میدان و جمعاً ۸ اشکال در یک دستور انشعاب است.

تغییرات نرم‌افزاری اعمال شده در این مکانیزم، به دلیل افزودن دستورهای جدید به مجموعهٔ دستورهای کاربر، کاهش کارآیی ریزپردازنده را در پی خواهد داشت که در بخش ارزیابی به آن خواهیم پرداخت.

تغییر سخت‌افزاری برای پیاده سازی مکانیزم BMBC: این تغییرات شامل افزودن واحدهایی، مطابق شکل ۴، به خط لولهٔ ریزپردازنده جهت استفاده از دستور l.bmbc و تصحیح اشکال در دستور انشعاب بعد از دستور l.bmbc است.



شکل ۴- پیاده سازی مکانیزم BMBC در خط لولهٔ ریزپردازنده

واحد واکشی دستور از حافظه دستور، Fetch، با استفاده از مداری که به آن اضافه شده است دستور l.bmbc را تشخیص می‌دهد. اگر دستور واکشی شده از حافظه دستور l.bmbc باشد، Fetch به CDR، که عملکرد آن شبیه یک ثبات است، اطلاع می‌دهد تا این واحد دستور l.bmbc را در سیکل بعد ذخیره کند. در سیکل بعد این دستور از واحد Fetch خارج شده به سمت واحد رمزگشایی، Decode، می‌رود. با افزودن واحدهای جدید عملکرد عادی ریزپردازنده نباید مختل شود، لذا بعد از Fetch یک مالتی‌پلکسر (MUX) اضافه شده است. MUX با توجه به محتویات Fetch تعیین می‌کند که کدام ورودی به خروجی وصل شود. در صورتی که محتوای Fetch چیزی غیر از دستور انشعاب باشد MUX ورودی صفر را انتخاب می‌کند، بدین ترتیب محتوای Fetch، دستور l.bmbc، مستقیماً به Decode منتقل می‌شود و با استفاده از تغییراتی که در واحد Decode داده شده است از این مرحله به بعد دستور l.bmbc به عنوان دستور NOP در خط لوله قلمداد می‌شود. خروجی دیگری نیز به واحد Fetch اضافه شده است، این خروجی به ورودی واحد CDR وصل شده است. به این وسیله اگر دستوری که از Fetch به سمت Decode می‌رود دستور l.bmbc باشد، دستور در CDR ذخیره می‌شود. با توجه به مکانیزم درج دستور l.bmbc در صورت عدم بروز وقفه، دستور بعدی که واکشی شده، دستور انشعاب است. لذا در سیکل سوم، هنگام ورود دستور انشعاب به واحد Decode باید در صورت لزوم تصحیح انجام گیرد. واحد CL که به همین منظور طراحی شده است یک مدار ترکیبی است که دو دسته ورودی دارد:

۱) خروجی واحد CDR (اطلاعات دستور l.bmbc)

۲) دستور انشعاب (دستور بعد از دستور l.bmbc)

خروجی واحد CDR در سیکل دوم یعنی همزمان با ورود دستور l.bmbc به واحد رمزگشایی، تولید می‌شود و همچنان ثابت نگه‌داشته می‌شود، لذا برای تولید آن زمانی صرف نخواهد شد. ورودی دوم واحد CL هم که از واحد Fetch تأمین می‌شود و در سیکل سوم در دسترس است و در اختیار واحد CL قرار داده می‌شود و خروجی CL، یعنی دستور انشعاب اصلاح شده با استفاده از مکانیزم همینگ، به دلیل ترکیبی بودن مدار در همین سیکل در دسترس است. انشعابی بودن دستور

سیکل دوم در واحد Fetch سبب می‌شود که در سیکل سوم واحد MUX، ورودی یک را به خروجی متصل کند، لذا دستور انشعابی اصلاح شده به واحد Decode منتقل خواهد شد.

لازم به ذکر است که تغییرات سخت‌افزاری در این مکانیزم موجب کاهش کارآیی نخواهد شد چرا که در مجموعهٔ واحدهای اضافه شده، تنها CDR یک مدار ترتیبی (ثبات) است و ذخیرهٔ دستور l.bmbc در این ثبات به موازات ورود و ذخیرهٔ همین دستور در ثبات ورودی واحد Decode صورت می‌گیرد لذا سربار زمان اجرا به ریزپردازنده تحمیل نمی‌شود. در مورد دستورهای انشعاب نیز به دلیل ترکیبی بودن مدار واحد CL، کاهش کارآیی نخواهیم داشت.

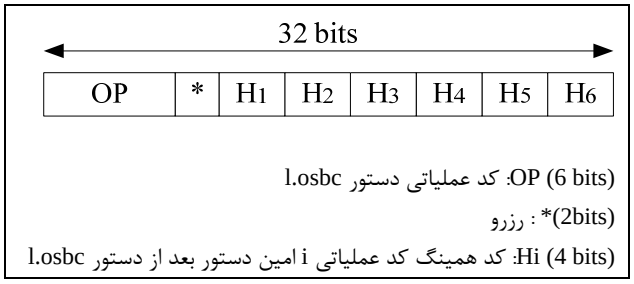
در مواردی چون بروز وقفه یا استثنا بعد از دستور l.bmbc ممکن است دستور انشعاب وارد خط لوله نشود، در این حالت نباید دستور بعدی که وارد خط لوله می‌شود به غلط اصلاح شود. در این مکانیزم تا رسیدن دستور l.bmbc بعدی، محتوای l.bmbc قبلی در ثبات CDR باقی می‌ماند. لذا بعد از بازگشت از وقفه یا استثنا، به محض تشخیص وجود دستور انشعاب در واحد واکشی، مجدداً فرآیند بررسی دستور انشعاب و اصلاح آن از سر گرفته می‌شود.

۴-۲- مکانیزم دوم (OSBC) از روش PFC

این مکانیزم جهت کشف و تصحیح اشکالات SBF در کد عملیاتی کل دستورهای برنامه، طراحی شده است. این مکانیزم قادر به کشف و تصحیح اشکالات نوع BIF و BDF و NBOMF است. لازم به ذکر است که این مکانیزم شامل تصحیح اشکال کد عملیاتی دستور l.bmbc نیز هست.

تغییر نرم‌افزاری برای پیاده سازی مکانیزم OSBC: این مکانیزم نیاز به ایجاد تغییر در بارکاری مورد نظر دارد. مکانیزم OSBC یک دستور طراحی شدهٔ خاص، دستور l.osbc، را هر شش خط یک بار در برنامه درج می‌کند تا بتواند قبل از تبدیل اشکال احتمالی موجود در کد عملیاتی در شش دستور بعدی به خطا، آن را تصحیح کند. این دستور حاوی ۶ میدان ۴ بیتی است. این چهار بیت، در واقع بیت‌های توازن در ساختار کد همینگ (۶ و ۱۰) است که در چهار مرحله پردازش به برنامه کاربر افزوده شده است. در واقع چهار مرحله‌ای بودن این تغییر به دلیل ادغام آن با تغییرات مکانیزم قبلی است. در ابتدا تمام تغییرات لازم به برنامه اسمبلی اعمال می‌شود و دستورهای جدید به آن افزوده می‌شوند و تازه بعد از این مرحله است که می‌توان تغییرات بعدی را به کد هگزادسیمال برنامه، حاصل از پردازش برنامه اسمبلی اعمال کرد. چون هر مکانیزم یک تغییر نرم‌افزاری دارد، در نهایت ۲ پس‌پردازشگر برای اعمال تغییرات به برنامه اسمبلی و دو پس‌پردازشگر برای اعمال تغییرات به کد هگزادسیمال مورد نیاز است. دلیل ادغام نشدن دو به دوی این پس‌پردازشگرها، تست ریزپردازنده در هر مرحله به طور جداگانه بوده است. ابتدا پس‌پردازشگر اول در مکانیزم BMBC برنامه اسمبلی حاصل از برنامه کاربر را به عنوان ورودی پذیرفته و دستورهای خام l.bmbc را قبل از هر یک از دستورهای انشعاب درج می‌کند، بعد از این مرحله پس‌پردازشگر نخست مکانیزم OSBC خروجی قبلی را به عنوان ورودی پذیرفته و هر ۶ دستور یکبار دستور l.osbc را درج می‌کند. در مرحلهٔ بعد پس از پردازش این برنامه توسط ریزپردازنده، کد هگزادسیمال دستورهای برنامه حاصل می‌شود.

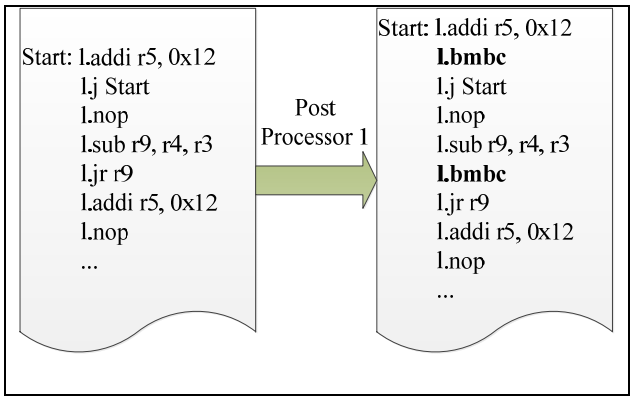
در این مرحله هنوز کدهای همینگ درج نشده‌اند و چون ریزپردازنده از قبل هر دوی این دستورها را می‌شناخته، کد عملیاتی آن‌ها را درج کرده ولی مابقی بیت‌های دستور را با مقادیر رزرو که در این ریزپردازنده "صفر" می‌باشند، پر کرده است. لازم به ذکر است که یکی از راه‌های تولید کد هگزادسیمال دستورها که حافظهٔ دستور ریزپردازنده با آن پر می‌شود، پردازش برنامه اسمبلی کاربر توسط ریزپردازنده است. در مرحلهٔ بعد با کمک پس‌پردازشگر دوم مکانیزم BMBC، برنامه تحلیل شده و کد همینگ دستورهای انشعاب محاسبه و طبق ساختار شکل ۲ در کد هگزادسیمال دستور l.bmbc درج می‌گردد. در آخر هم پس‌پردازشگر دوم مکانیزم OSBC، کدهای همینگ کدهای عملیاتی دستورها (۶ دستوری که بعد از هر دستور l.osbc آمده است) را محاسبه کرده و طبق ساختار شکل ۵ این کدها را در کد هگزادسیمال دستور l.osbc درج خواهد کرد.



شکل ۵- ساختار کلی دستور l.osbc

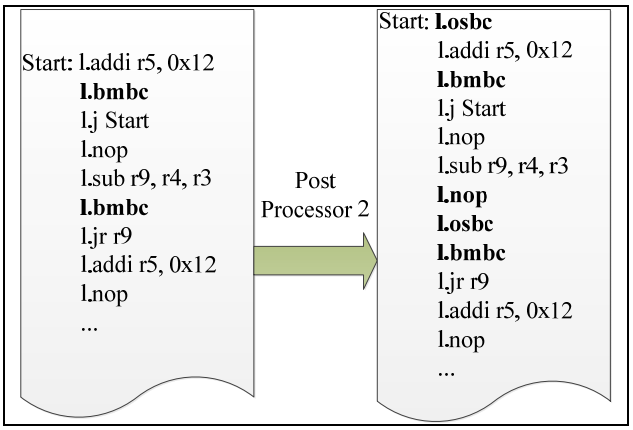
در واقع در ساختار این دستور ۶ عملوند بلافصل H_i وجود دارد. هر عملوند ۴ بیتی H_i، بیت‌های توازن کد همینگ کد عملیاتی i امین دستور بعد از دستور l.osbc است. H_i دارای ۴ بیت است که حاصل از کدهای عملیاتی ۶ بیتی هر دستور است. تصحیح کد عملیاتی دستورها باید از همان ابتدای برنامه آغاز شود لذا در ابتدای برنامه دستور l.osbc در محل اولین دستور برنامهٔ بارکاری درج می‌شود و پس از آن هر ۶ دستور یکبار این روند ادامه می‌یابد. پس از پرش به بخش دیگری از برنامهٔ بار کاری، توالی ۶ دستوری به هم خواهد خورد. این دستور انشعاب ممکن است در بین ۶ دستور باشد یا دقیقاً ششمین دستور باشد ولی در هر صورت توالی به هم خواهد خورد و ممکن است کد عملیاتی دستورهای بعدی به غلط اصلاح شوند لذا برای جلوگیری از این تداخل در مقاصد انشعاب هم این دستور یکبار دیگر درج می‌شود و تصحیح از نو برای مجموعهٔ دستورهای جدید آغاز می‌شود. لازم به ذکر است که تشخیص مقاصد پرش از روی برنامهٔ کاربر به سادگی ممکن است، چرا که هر برچسب در برنامهٔ اسمبلی یک مقصد پرش بالقوه است و با استفاده از این نکته در تمامی این مکان‌ها، دستور l.osbc مجدداً درج می‌شود.

اگر در توالی ۶ دستور آخرین دستور یک دستور l.bmbc باشد، با درج l.osbc بعد از آن، در فرآیند اصلاح دستور انشعاب اشکال ایجاد خواهد شد. پس‌پردازشگر مکانیزم OSBC طوری تنظیم شده است که در این موارد به جای دستور ششم، دستور NOP درج می‌کند و l.bmbc را به عنوان دستور اول از توالی بعدی در نظر می‌گیرد. تغییرات اعمال شده بر قطعه‌ای از یک برنامهٔ بارکاری توسط پس-پردازشگر اول از مکانیزم OSBC در شکل ۶ نشان داده شده است.



شکل ۶- تغییر در برنامهٔ بارکاری حاصل از پس‌پردازشگر اول در مکانیزم OSBC

همانطور که مشاهده می‌شود، در شکل ۶ پس‌پردازشگر اول دستور l.bmbc را قبل از دستورهای انشعاب (l.j و l.jr) درج کرده است. در شکل ۷ پس‌پردازشگر دوم از مکانیزم OSBC دستورهای l.osbc را درج می‌کند. به دلیل اینکه مکانیزم OSBC بر اساس تعداد دستورها تنظیم شده است، ابتدا باید دستورهای l.bmbc درج شوند سپس مکانیزم OSBC دستورهای l.osbc را درج می‌کند.



شکل ۷- تغییر در برنامهٔ بارکاری حاصل از پس‌پردازشگر دوم در مکانیزم OSBC

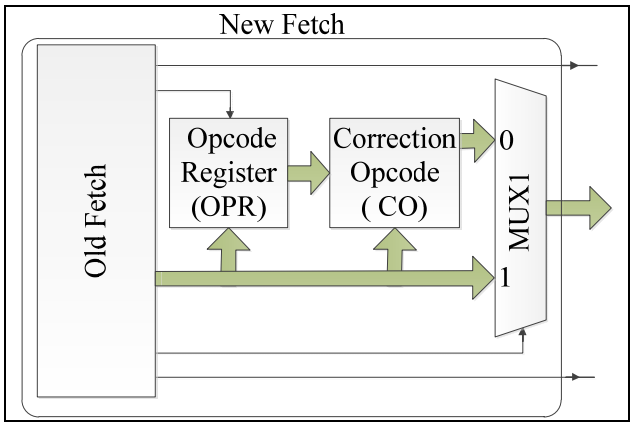
در شکل ۷ پس‌پردازشگر دوم از مکانیزم OSBC در زمان رسیدن به برچسب Start یکبار دستور l.osbc را درج می‌کند. بعد از این برچسب، ششمین دستور، یک دستور l.bmbc است. همانطور که گفته شد دستور l.bmbc برای تصحیح دستورهای انشعاب در نظر گرفته شده و قبل از این دستورها درج می‌شود و مکانیزم OSBC با درج دستور l.osbc مانع از حصول این هدف خواهد شود. لذا در اینطور موارد، پس‌پردازشگر دوم از مکانیزم OSBC دستور NOP (l.nop) را به عنوان دستور ششم در لیست دستورها ثبت کرده و l.bmbc را اولین دستور از گروه ۶ دستوری بعدی به حساب می‌آورد.

تغییر سخت‌افزاری برای پیاده‌سازی مکانیزم OSBC: این تغییرات شامل افزودن واحدهایی در خط لوله پیش از واحدهای تعبیه شده در مکانیزم BMBC است تا اشکال احتمالی کد عملیاتی دستور l.bmbc قبل از اجرای مکانیزم BMBC در خط لوله تصحیح شود.

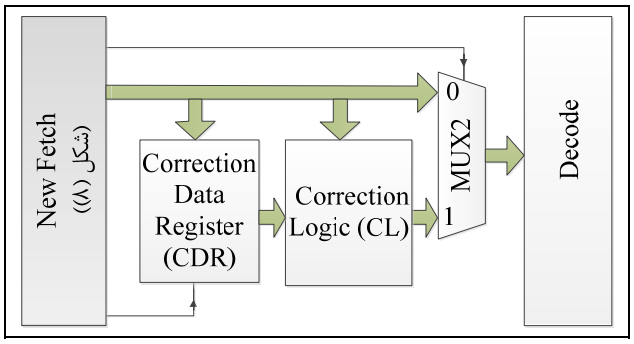
واحد Fetch، در پیاده‌سازی این مکانیزم نسبت به مکانیزم BMBC، نیاز به تغییرات بیشتری دارد چرا که علاوه بر اینکه باید تغییرات سخت‌افزاری لازم برای مکانیزم OSBC بر آن اعمال شود، باید این تغییرات با تغییرات سخت‌افزاری لازم برای مکانیزم BMBC هم هماهنگ شود. برای ایجاد این هماهنگی باید واحد Fetch جدیدی تعریف شود.

همانطور که اشاره شد مکانیزم OSBC کد عملیاتی دستورها را تصحیح می‌کند و از طرفی در شکل ۴ واحدهای CDR بر اساس کد عملیاتی دستور تصمیم بر ذخیره کردن یا نادیده گرفتن دستور می‌گیرد و نیز واحد MUX بر اساس کد عملیاتی دستور، ورودی مورد نظر را انتخاب می‌کند. لذا برای انتخاب صحیح در این واحدها می‌بایست ابتدا کد عملیاتی تصحیح شود و سپس در اختیار این واحدها قرار گیرد. بنابراین تغییرات مکانیزم OSBC قبل از مکانیزم BMBC قرار می‌گیرد و به این ترتیب واحد واکشی جدید شامل واحد Fetch قبلی و واحد تصحیح کد عملیاتی است. در شکل ۸ این واحد Fetch جدید (New Fetch) نشان داده شده است.

در شکل ۸ ارتباط واحد New Fetch با واحدهای بعد از خودش خارج از بلاک نشان داده شده است. بعد از طراحی، این واحد جدید به جای واحد Fetch Old جاگذاری شد (شکل ۹).



شکل ۸- واحد واکشی جدید (New Fetch)



شکل ۹- سخت‌افزار مکانیزم OSBC

عملکرد این مدار را می‌توان به این نحو توضیح داد که واحد Old Fetch با استفاده از مداری که به آن اضافه شده است دستور l.osbc را تشخیص می‌دهد. اگر دستور واکشی شده از حافظهٔ دستور l.osbc باشد، Old Fetch به OPR، که عملکرد آن شبیه یک ثبات است، اطلاع می‌دهد تا این واحد دستور l.osbc را در سیکل بعد ذخیره کند. برای این منظور خروجی دیگری برای انتقال دستور از واحد Old Fetch به بخش‌های دیگر، به این واحد افزوده شده است. این خروجی به ورودی واحد OPR وصل شده است و در صورتی که دستور موجود در واحد Old Fetch، l.osbc باشد، OPR این دستور را از این ورودی برداشته و ذخیره می‌کند. یک شمارنده در واحد OPR طراحی شده است تا از این مرحله به بعد شروع به شمارش کرده و توالی شش دستور یکبار را حفظ کند. در سیکل دوم این دستور از واحد Old Fetch خارج می‌شود و بدون هیچ تغییری باید به سمت واحد Decode برود. لذا بعد از واحد Old Fetch، یک مالتی‌پلکسر اضافه شده است که با توجه به خروجی جدیدی که به واحد Old Fetch اضافه شده است تعیین می‌کند که کدام یک از ورودی‌هایش به خروجی‌اش متصل شود.

MUX1 در صورتی که محتوای Old Fetch دستور l.osbc باشد ورودی یک را انتخاب می‌کند. این دستور مسیر خود را ادامه می‌دهد و چون دستور از نوع l.bmbc نیست خروجی واحد Fetch جدید برای واحد CDR فعال نمی‌شود و این واحد دستور l.osbc را ذخیره نمی‌کند. در ادامهٔ مسیر، MUX2، که همان MUX در شکل ۴ است با توجه به خروجی واحد New Fetch، ورودی صفر خود را برای اتصال به خروجی انتخاب می‌کند چرا که همانطور که اشاره شد این مالتی‌پلکسر تنها در صورتی که دستور انشعاب در واحد Fetch جدید باشد، ورودی ۱ را برای اصلاح احتمالی دستور انشعاب انتخاب خواهد کرد. بدین ترتیب دستور l.osbc در سیکل دوم بدون هیچ تغییری به واحد Decode وارد می‌شود و با استفاده از تغییراتی که در واحد Decode داده شده است از این مرحله به بعد به عنوان دستور NOP در خط لوله قلمداد می‌شود.

از این سیکل به بعد شمارندهٔ واحد OPR شروع به شمارش خواهد کرد. در سیکل دوم دستور جدیدی از حافظه دستور واکشی شده، وارد واحد Old Fetch خواهد شد. برای این دستور دو احتمال وجود دارد، این دستور یا l.osbc است یا l.osbc نیست. اگر این دستور l.osbc نباشد، باید کد عملیاتی آن اصلاح شود. واحد CO که به همین منظور طراحی شده است یک مدار ترکیبی است که دو دسته ورودی دارد:

۱) خروجی واحد OPR (اطلاعات دستور l.osbc)

۲) دستورهای عادی برنامه کاربر (دستوری غیر از l.osbc)

خروجی واحد OPR در سیکل دوم یعنی همزمان با ورود دستور l.osbc به واحد Decode، تولید می‌شود و همچنان ثابت نگه‌داشته می‌شود، لذا برای تولید آن زمانی صرف نخواهد شد. ورودی دوم واحد CO هم که از واحد Old Fetch تأمین می‌شود، در سیکل سوم در دسترس است و در اختیار واحد CO قرار داده می‌شود. خروجی CO، یعنی دستوری که کد عملیاتی آن با استفاده از مکانیزم کد همینگ اصلاح شده است، به دلیل ترکیبی بودن مدار در همین سیکل در دسترس است. به دلیل اینکه این دستور، l.osbc نیست لذا اصلاحات احتمالی بر روی آن صورت گرفته و نباید نسخهٔ بدون تغییر آن به واحد Decode وارد شود. این مورد سبب می‌گردد که در سیکل سوم واحد MUX2، ورودی صفر را به خروجی متصل کند. این دستور مسیر خود را ادامه خواهد داد و در خروج از واحد New Fetch دو احتمال برای این دستور در نظر گرفته می‌شود: این دستور یا l.bmbc است یا دستوری غیرانشعابی است.

این دستور نمی‌تواند دستوری انشعابی باشد، چرا که طبق مکانیزم‌های پیاده شده به کمک پس‌پردازشگر مکانیزم BMBC، قبل از هر دستور انشعاب یک دستور l.bmbc قرار داده شده است، در حالی که قبل از این دستور، دستور l.osbc قرار داشت. اگر این دستور، l.bmbc باشد، واحد New Fetch به CDR اطلاع خواهد داد و این واحد محتویات خود را با خروجی واحد New Fetch به روز می‌کند و MUX2 نیز با توجه به انتخاب کنندهٔ آن که از واحد New Fetch می‌آید متوجه می‌شود که باید ورودی صفر را به خروجی‌اش وصل کند. از اینجا به بعد تمامی مراحل شبیه به آنچه که در تغییرات سخت‌افزاری مکانیزم BMBC توضیح داده شد پیش خواهد رفت. اگر دستور خروجی از واحد New Fetch، یک دستور غیرانشعابی باشد (l.bmbc هم نباشد) بدون تغییر و نیز بدون اینکه محتوای CDR را تغییر دهد از ورودی صفر MUX2 به واحد Decode منتقل می‌شود.

حال اگر دستور واکشی شده از حافظه در سیکل دوم دستور l.osbc باشد، شرایط اندکی فرق خواهد کرد. از آنجایی که پس‌پردازشگر نخست از مکانیزم OSBC طوری تنظیم شده است که هر ۶ دستور یکبار، دستور l.osbc را درج کند، مشاهدهٔ دو دستور l.osbc پشت سرهم منطقی به نظر نمی‌رسد. اما اگر دستور l.osbc اول در delay slot دستور انشعاب قبل از آن واقع شده باشد این امر ممکن خواهد بود. delay slot برای ریزپردازندهٔ OpenRisc معمولاً دستور NOP خواهد بود. اما در اینجا چون دستور l.osbc هر شش دستور یکبار درج شده است، ممکن است این توالی به هم بریزد و دستور l.osbc در delay slot قرار بگیرد. در هر صورت با مشاهدهٔ مجدد دستور l.osbc همهٔ فرآیندها از نو می‌شوند و شمارندهٔ واحد OPR نیز مجدداً صفر خواهد شد.

در صورتی که دستور واکشی شده از حافظه دستور l.osbc نباشد توالی ۶ دستور ادامه خواهد یافت. در دو حالت توالی به انتها خواهد رسید:

- ۶ سیکل زمانی سپری شود و شش دستور متوالی فراخوانی شده و اصلاح شوند.
- برنامه به محل یک برچسب برسد.

حالت دوم، در واقع با توجه به قرارداد قبلی در مورد تغییرات نرم‌افزاری حاصل از پس‌پردازشگر اول مکانیزم OSBC همان حالتی است که در طول توالی ۶ دستور با دستور l.osbc برخورد می‌کنیم که موجب از نو شدن عملیات‌های مربوط به مکانیزم دوم خواهد شد.

اگر در توالی ۶ دستور، یک دستور انشعاب وجود داشته باشد، تمام مراحل مانند دستور بعد از l.osbc خواهد بود با این تفاوت که در MUX2 ورودی ۱ انتخاب خواهد شد.

SBF ممکن است باعث بروز اشکال در کد عملیاتی دستور l.osbc شده و استثنای دستورالعمل غیر مجاز را فعال کند (بروز اشکال SBF در کد عملیاتی دستور l.bmbc توسط مکانیزم OSBC کشف و تصحیح می‌شود). برای حفاظت کد عملیاتی این دستور نیز ساختار جدیدی طراحی شده است. با فرض بروز خطای یک‌بیتی در این کد ۶ بیتی، کد عملیاتی میتواند به ۶ کد عملیاتی دیگر تبدیل شود. از طرفی به خاطر رعایت حداقل فاصله همینگ ۲ در کد عملیاتی این دستور، هیچ یک از این کدها، از کدهای عملیاتی مجاز نیستند. از سوی دیگر اگر عملیات کشف و تصحیح اشکال کد عملیاتی این دستور را در Decode قرار دهیم، مطمئن خواهیم بود که تمام کدهای عملیاتی تصحیح شده‌اند و تنها امکان بروز اشکال در کد این دستور وجود دارد.

به محض شناسایی یک دستور ناشناس توسط Decode، این واحد سیگنال خطا را فعال می‌کند. واحد CO کد عملیاتی اصلی (قبل از تصحیح) هر دستوری که در میدان دیدش قرار بگیرد، حداقل برای یک سیکل زمانی حفظ می‌کند (اعم از l.osbc یا غیر از آن). واحد OPR کد عملیاتی مرجع را از واحد CO برداشته با ۶ دستور یاد شده مقایسه می‌کند (دلیل استفاده نکردن از کد عملیاتی دستور واحد Decode آن است که کدعملیاتی این دستور به علت گذر از مکانیزم OSBC احتمالاً به اشتباه تغییر یافته است). در صورت برابری کد مرجع با یکی از این دستورها، دستوری که SBF بر آن تأثیر گذاشته یک دستور l.osbc می‌باشد. به این ترتیب این دستور به عنوان دستور اصلاحی برای دستورهای بعدی در OPR قرار داده می‌شود (در اینجا فرض بر SBF بودن خطاست لذا سایر بیت‌های موجود در این دستور صحیح هستند). بعد از اصلاح این کد عملیاتی برای جلوگیری از انتشار کد غلط قبلی در خط لوله، ورودی واحد اجرا هم به جای تأمین شدن از واحد Decode، این بار از واحد OPR تأمین می‌شود.

لازم به ذکر است که کاهش کارآیی حاصل از تغییرات نرم‌افزاری و سخت‌افزاری این مکانیزم مانند مکانیزم BMBC است.

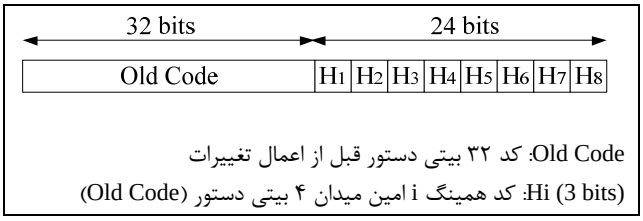
۵- پیاده‌سازی کد همینگ در حافظه (MFC)

برای مقایسه با روش قبلی، یک سیستم تصحیح خطا صرفاً در حافظهٔ ریزپردازنده ارائه و پیاده‌سازی شده است. به این ترتیب در هر ورود/خروج داده یا دستور به/از حافظه، کدهای تصحیح، تولید/وارسی شده و در صورت لزوم تصحیح شده و سپس به ریزپردازنده تحویل داده می‌شوند.

MFC از کد همینگ برای حفاظت داده و دستور استفاده می‌کند. این کدها در انتهای هر کلمه از حافظه قرار می‌گیرند. بنابراین در MFC تعداد کلمات حافظه تغییری نمی‌کند بلکه طول کلمات بیشتر می‌شود. منظور از تغییرات سخت‌افزاری در واقع افزودن واحدهایی در ورودی و خروجی حافظه است که اعمال مربوط به رمزگذاری داده و دستور برای ذخیره در حافظه و رمزگشایی داده و دستور فراخوانی شده از حافظه را انجام می‌دهد. این تغییرات در کد Verilog حافظهٔ ریزپردازنده اعمال شده است. این روش، به ازای هر دستور ۳۲ بیتی از حافظهٔ ریزپردازندهٔ اولیه، یک دستور ۵۶ بیتی را در هر یک از کلمات حافظهٔ دستور ریزپردازنده درج می‌کند.

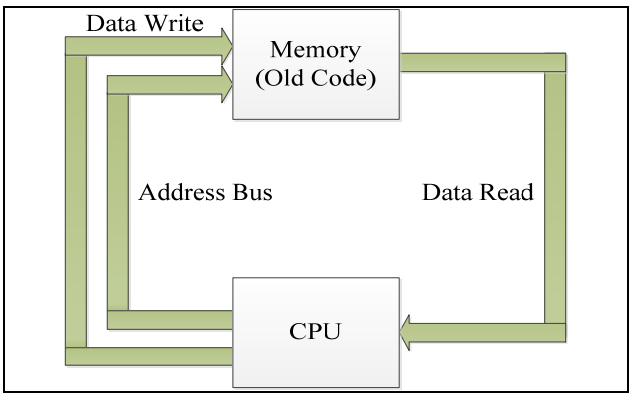
این دستور، محتوای ۳۲ بیتی قبلی را در ابتدای کلمهٔ جدید درج می‌کند و در ادامه حاوی ۸ میدان ۳ بیتی (۲۴ بیت) است که هر یک از این سه بیت، در واقع

بیت‌های توازن در ساختار کد همینگ (۴و ۷) حاصل از میدان‌های ۴ بیتی دستور اولیه است که در یک مرحله پردازش به برنامهٔ کاربر افزوده شده است. ساختار کلمات حافظهٔ ریزپردازنده پس از اعمال این تغییرات در شکل ۱۰ نشان داده شده است. در این حالت عرض حافظه بیش‌تر از ۳۲ بیت (۵۶ بیت) خواهد بود چرا که کدهای تصحیح خطا نیز به آن افزوده شده است.



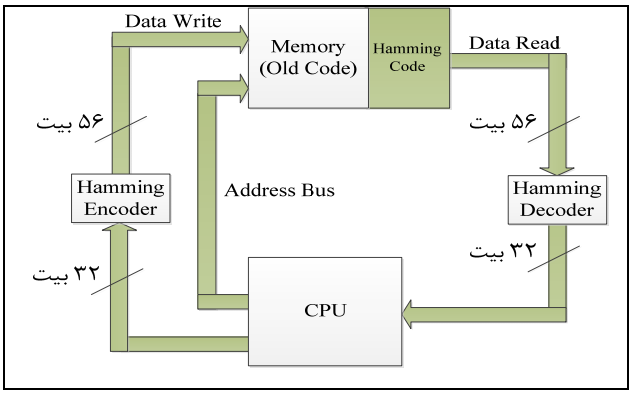
شکل ۱۰- ساختار کلمات حافظه بعد از اعمال روش MFC

تغییرات سخت‌افزاری روش MFC: در این مرحله از تغییرات باید مراحل کد گذاری و رمزگشایی هر دو پیاده‌سازی شود، چرا که علاوه بر واکشی دستورها از حافظه، که به واحد رمزگشایی نیاز دارد، نوشتن داده در حافظه و بازخوانی داده از حافظه به ترتیب به واحدهای رمزگذاری و رمزگشایی نیاز دارند. به این منظور به واحد حافظه، مداری ترکیبی اضافه شده است که ارتباط با حافظه را از رمزگذار/رمزگشا عبور خواهد داد. در شکل ۱۱ مسیر معمول تبادل اطلاعات بین حافظه و ریزپردازنده نشان داده شده است.



شکل ۱۱- چگونگی ارتباط حافظه با ریزپردازنده قبل از اعمال روش MFC

پس از پیاده‌سازی MFC نحوهٔ ارتباط حافظه و ریزپردازنده به صورت ساختار نشان داده شده در شکل ۱۲ خواهد بود.



شکل ۱۲- نحوهٔ ارتباط حافظه با ریزپردازنده بعد از پیاده‌سازی روش MFC

در ماژول ارتباط با حافظه، زمان نوشتن داده در حافظه، ابتدا این داده رمزگذاری می‌شود و سپس داده رمزگذاری شده از همان مسیر قبلی در حافظهٔ داده درج می‌شود. همچنین در زمان واکشی داده (یا فراخوانی دستور)، این داده (یا دستور) ابتدا رمزگشایی شده و سپس به ریزپردازنده تحویل داده می‌شود چرا که برای سایر بخش‌های ریزپردازنده صرفاً کد اصلی دستور قابل پردازش است. به دلیل افزایش طول کلمات حافظه، طول گذرگاه‌های نوشتن در حافظه و خواندن از حافظه هم باید افزایش پیدا کند و از ۳۲ بیت به ۵۶ بیت تغییر یابد.

لازم به ذکر است که از مزایای روش MFC، همراه بودن کد ارسالی با کد همینگ تصحیح کننده است. لذا رخداد استثنا، وقفه یا انشعاب، مشکلی در پیوستگی عملیات ایجاد نخواهد کرد، چرا که در فراخوانی هر دستور یا واکشی هر داده، دستور و داده با کد همینگ مربوط به خود فراخوانی خواهند شد. مزیت دیگر این روش آن است که تغییری در برنامهٔ کاربر اعمال نمی‌کند به عبارت پیاده‌سازی این روش نیازی به تغییرات نرم‌افزاری ندارد. این روش به دلیل ایجاد افزونگی داده در حافظه، سربار سخت‌افزار بیشتری نسبت به روش PFC دارد. از سوی دیگر به دلیل آنکه ماژول‌های کدگذاری و رمزگشایی در ریزپردازنده بر هر کلمه از حافظه اعمال می‌شود، سربار توان مصرفی بالایی به ریزپردازنده تحمیل خواهد شد.

۶- ارزیابی روش های ارائه شده

از آنجا که ایدهٔ اصلی این روش‌ها عمومی بوده و بر قابلیت خاصی تکیه ندارد، با تغییرات مختصر بر روی هر پردازنده با کد توصیف سخت‌افزار آزاد، از قبیل OpenRISC و SPARC قابل پیاده‌سازی است. نظر به اینکه روش IMCFEDC [8] نزدیکترین روش به مکانیزم BMBC است و آن روش بر روی ریزپردازندهٔ OpenRISC پیاده شده است، به منظور مقایسه بهتر با آن روش، روش‌های PFC و MFC نیز بر روی ریزپردازندهٔ OpenRISC پیاده‌سازی شده‌اند.

صحت عمل این روش‌ها توسط شبیه‌سازهای Modelsim و Icarus ارزیابی گردیده است و سپس در مرحلهٔ بعد میزان توان مصرفی و سخت‌افزار بکار رفته برای پیاده‌سازی طرح با کمک ابزار Synopsys و با استفاده از تکنولوژی TSMC 0.18um محاسبه شده است. در این فصل پارامترهای مختلف محاسبه شده در فرآیندهای شبیه‌سازی و سنتز بررسی خواهد شد.

پوشش تصحیح اشکال: پوشش تصحیح اشکال که یکی از پارامترهای مهم برای ارزیابی روش‌های تصحیح خطا است، نشان دهندهٔ نسبت اشکالات کشف و تصحیح شده به کل اشکالات موجود برحسب مدل اشکال ارائه شده است [۱۶]. این پارامتر در این مقاله به صورت تحلیلی محاسبه شده است.

سربار سخت‌افزار: سربار سخت‌افزار در مورد روش‌هایی که نیازمند تغییر در سخت‌افزار سیستم هستند مورد توجه قرار می‌گیرد. در طراحی‌هایی که میزان سخت‌افزار مصرفی و در نتیجه جنبهٔ اقتصادی طرح اهمیت دارد، پایین بودن این مقدار می‌تواند یک مزیت باشد. در اینجا هر دو روش ارائه شده شامل تغییرات سخت‌افزاری بوده‌اند، بنابراین برای هر دو روش سربار سخت‌افزار غیر صفر خواهیم داشت.

سربار توان مصرفی: میزان توان یا انرژی مصرفی در گذشته کمتر مورد توجه قرار می‌گرفته، اما اخیراً با ورود روش‌هایی با توان مصرفی کمتر[۳۴] اهمیت بیشتری پیدا کرده است. از این رو سربار توان مصرفی می‌تواند معیاری برای انتخاب یک روش مناسب، محسوب شود.

سربار حجم حافظه: اکثر روش‌های تحمل‌پذیری اشکال[۳۵] نیازمند افزونگی در سیستم هستند. این افزونگی ممکن است در سخت‌افزار ریزپردازنده یا در حافظهٔ ریزپردازنده باشد. افزونگی در حافظه ممکن است بر اثر افزودن دستورهایی به مجموعه دستورهای کاربر (مانند روش PFC) یا ممکن است بر اثر افزودن داده

به مجموعهٔ داده‌های ذخیره شده در حافظه باشد (مانند روش‌هایی که بر اساس پیاده‌سازی کد Reed-solomon در حافظهٔ دستور بنا شده‌اند یا مانند روش MFC). در این رابطه پارامتر دیگری برای ارزیابی طرح‌های مختلف و بررسی مزایا و معایب آن‌ها با نام سربار حجم حافظه تعریف شده است.

سربار زمان اجرای برنامه: در مورد آن دسته از روش‌های کشف خطای روند اجرا که نیازمند افزودن دستورهایی به برنامه کاربر هستند پارامتر با اهمیت دیگر، سربار زمان اجرای برنامه است. اجرا شدن دستورهای افزونه درون ریزپردازنده باعث افزایش زمان اجرای برنامه می‌شود که این افزایش تحت عنوان سربار زمان اجرا اندازه‌گیری می‌شود و نشان‌دهندهٔ درصد افزایش زمان اجرای برنامه نسبت به حالت اولیه، بدون استفاده از مکانیزم کشف خطا یا اشکال، است [۱۶]. این نکته قابل ذکر است که صرفاً افزایش دستورها سبب افزایش زمان اجرا نمی‌گردد. برای مثال روش‌های طراحی پردازندهٔ تحمل‌پذیر خطا که بر اساس کد Reed-solomon پیاده‌سازی می‌شوند، به دلیل فرآیندهای طولانی رمزگذاری و رمزگشایی دستورهای کاربر، سربار زمان اجرای بالایی ایجاد می‌کنند. سربار زمان اجرا معیاری است برای سنجش کارایی سیستم که به بیانی به زمان مورد نیاز برای تولید خروجی مرتبط است.

در بخش‌هایی که در ادامه خواهد آمد، پارامترهای ذکر شده در بالا برای هر دو روش PFC و MFC محاسبه و ارائه شده است.

۶-۱- نتایج ارزیابی روش PFC

در جدول ۱ پوشش تصحیح اشکال روش PFC ارائه شده است. هر دو مکانیزم BMBC و OSBC یک اشکال را ۱۰۰٪ تصحیح می‌کنند اما مزیت BMBC در آن است که این مکانیزم قادر به تصحیح چند اشکال همزمان است که امروزه از اهمیت بالایی برخوردار شده است [۱].

برای محاسبهٔ مقادیر ارائه شده در جدول ۱ از روش تحلیلی استفاده شده است:

$$\text{پوشش کشف و تصحیح خطا} = \frac{\text{تعداد محل‌های بروز خطای قابل کشف و تصحیح}}{\text{تعداد محل‌های بروز خطا}}$$

به این ترتیب برای محاسبهٔ احتمال کشف تا ۲ بیت اشکال همزمان این رابطه به این ترتیب خواهد بود:

$$\frac{\binom{8}{1}\times\binom{4}{1}}{\binom{32}{1}} + \frac{\binom{8}{2}\times\binom{4}{1}\times\binom{4}{1}}{\binom{32}{2}} = P(<=2)$$

همانطوریکه در جدول ۱ آمده است، درصد پوشش تصحیح اشکال برای اشکال با تعداد بیت کم، بسیار بالا است. به عنوان مثال پوشش تصحیح اشکال برای ۱ بیت اشکال ۱۰۰٪ است. با افزایش تعداد بیت اشکال، پوشش تصحیح اشکال کم می‌شود. ولی این واقعیت مهم است که احتمال رخداد اشکال با بیت‌های کم (مثلاً یک بیت اشکال) بالاتر از احتمال رخداد اشکال با بیت‌های بیشتر (مثلاً ۸ بیت اشکال) است (شکل ۱).

همچنین از جدول ۱ بدست می‌آید که روش PFC قادر به کشف اشکال NBPMF نمی‌باشد. این روش در مکانیزم OSBC کد عملیاتی تمام دستورها اعم از انشعابی و غیرانشعابی را محافظت می‌کند، اما قادر به محافظت از عملوند دستورهای غیرانشعابی نیست. دلیل آن است که مکانیزم BMBC فقط دستورهای انشعابی را محافظت می‌کند.

جدول ۱- پوشش تصحیح اشکال در روش PFC

حداکثر تعداد اشکال‌ها	پوشش تصحیح اشکال برحسب مدل اشکال	
	مکانیزم اول برای تصحیح BTMF, BDF	مکانیزم‌دوم برای تصحیح BIF, NBOMF
۱ بیت اشکال	٪۱۰۰	٪۱۰۰
تا۲ بیت اشکال	٪۹۰/۹۱	
تا۳ بیت اشکال	٪۷۴/۰۵	
تا۴ بیت اشکال	٪۵۳/۰۴	
تا۵ بیت اشکال	٪۳۲/۶۷	
تا۶ بیت اشکال	٪۱۶/۸۸	
تا۷ بیت اشکال	٪۵/۰۹	
تا۸ بیت اشکال	٪۲/۳۱	

ماژول‌های اضافه شده به ریزپردازنده در این روش، موجب افزایش حجم سخت‌افزاری سیستم خواهد شد و نیز سوییچینگ‌های مدارهای اضافه شده به نوبهٔ خود سبب افزایش توان مصرفی در ریزپردازنده خواهد شد. جهت محاسبهٔ سربارهای سخت‌افزار و سربار مصرف توان، ریزپردازنده با ابزار سنتز شرکت Synopsys (Synopsys Design Compiler) با تکنولوژی TSMC 0.18um سنتز شد. نتایج بدست آمده برای سربار سخت‌افزار در جدول ۲ آورده شده است.

جدول ۲- سربار سخت‌افزار روش PFC (برحسب میکرو متر مربع)

	مساحت بر حسب um^2		
ریزپردازندهٔ اولیه	ریزپردازنده با روش PFC	میزان سربار سخت‌افزار	درصد سربار سخت‌افزار
۱۸۸۲۰۸۰	۲۰۹۰۹۹۳	۲۰۸۹۱۳	٪۱۱/۱۰

از آنجایی که پارامتری چون توان مصرفی (و در ادامه پارامترهای بعدی) برای بارهای کاری مختلف مقادیر متفاوتی خواهد داشت، برای مقایسه با روش‌های پیشین بهتر است از برنامه‌های مشابهی استفاده کرد تا امکان مقایسه وجود داشته باشد. سه برنامهٔ مرتب‌سازی سریع و مرتب‌سازی حبابی و ضرب ماتریس از جمله برنامه‌های بارکاری هستند که در کارهای پیشین از آن‌ها استفاده شده است. برنامه‌های مرتب‌سازی سریع و مرتب‌سازی حبابی هر کدام با ۳۰ المان و برنامهٔ ضرب ماتریس برای دو ماتریس ۴×۴ در ریزپردازندهٔ OpenRISC اجرا شده‌اند. سربار توان مصرفی در هر یک از این برنامه‌ها در جدول ۳ آمده است. برنامهٔ مرتب‌سازی سریع به علت پیچیدگی بیشتر و تعداد بالای مقایسه برای مرتب‌سازی المان‌ها، سوییچینگ‌های بیشتری در مدار محاسباتی ریزپردازنده ایجاد خواهد کرد و مسلماً توان مصرفی بالایی داشته و سربار توان مصرفی بیشتری هم به ریزپردازنده تحمیل خواهد کرد.

سربار حجم حافظه برای بارهای کاری مرتب‌سازی سریع و مرتب‌سازی حبابی و ضرب ماتریس در جدول ۴ برای روش PFC محاسبه شده است. این سربار در اثر افزودن کدهای افزونه به بارکاری بوجود آمده است.

چون طول همهٔ دستورالعمل‌های ریزپردازنده OpenRISC، ۳۲ بیت می‌باشد، تعداد دستورهای افزونه معیار مناسبی از سربار حجم حافظه است. در جدول ۴ از شمارش تعداد دستورها استفاده شده است. به این منظور یک پس‌پردازشگر دیگر به زبان C++ طراحی شد که شمارش تعداد دستورها را قبل و بعد از اعمال روش به عهده داشت.

جدول ۳- سربار توان مصرفی روش PFC (بر حسب میلی وات)

بارکاری	ریزپردازندهٔ اولیه	ریزپردازنده با روش PFC	درصد سربار توان مصرفی
مرتب‌سازی سریع	۱۵۶/۷۳ mw	۱۶۳/۶۴ mw	٪۴/۴۱
مرتب‌سازی حبابی	۱۵۲/۸۹ mw	۱۵۸/۲۴ mw	٪۳/۵
ضرب ماتریس	۱۵۳/۳ mw	۱۵۸/۷۷ mw	٪۳/۵۷
میانگین	۱۵۴/۳۱ mw	۱۶۰/۲۲ mw	٪۳/۸۱

جدول ۴- سربار حجم حافظهٔ روش PFC

	مرتب‌سازی سریع	مرتب‌سازی حبابی	ضرب ماتریس
تعداد دستورات بارکاری مرجع	۲۸۵	۱۹۹	۲۳۹
تعداد دستورات انشعاب در بارکاری	۳۲	۲۰	۱۶
تعداد دستورات با اعمال روش	۳۸۴	۲۶۴	۳۰۵
سربار تعداد دستوات	۹۹	۶۵	۶۶
درصد سربار حجم حافظه	٪۳۴/۷۴	٪۳۲/۶۶	٪۲۷/۶۲
درصد میانگین سربار حافظه	٪۳۱/۶۷		

سربار حجم حافظه در مکانیزم BMBC به تعداد دستورهای انشعابی بستگی دارد و در مکانیزم OSBC سربار حجم حافظه به تعداد دستورهای برنامه بستگی دارد. با توجه به اینکه نسبت تعداد دستورهای انشعاب به کل دستورها در بارکاری مرتب‌سازی سریع بیشتر از مرتب‌سازی حبابی و ضرب ماتریس است، سربار حجم حافظه در آن نیز اندکی بیشتر است.

جدول ۵ مقادیر سربار زمان اجرای بارهای کاری را با اعمال روش PFC ارائه می‌دهد. در واقع در هر برنامهٔ بارکاری زمان خروج آخرین دستور از آخرین طبقهٔ خط لوله به عنوان زمان کلی اجرای یک برنامه در نظر گرفته شده است. نتایج این جدول با استفاده از نرم‌افزارهای Modelsim و GTKWave بدست آمده است. در واقع ریزپردازنده یکبار بدون اعمال تغییرات شبیه‌سازی می‌شود و مشخصات زمانی آن استخراج می‌شود. پس از آن با اعمال روش، بار دیگر شبیه‌سازی انجام شده و پارامتر زمان اجرا برای مقایسهٔ کارایی روش بدست می‌آید.

سربار زمان اجرا مانند سربار حجم حافظه به نسبت تعداد دستورهای اضافه شده در میان کل دستورها بستگی ندارد، بلکه به مقدار مطلق این پارامتر یعنی تعداد دستورهای افزوده شده وابسته است. در برنامه‌های بار کاری مرتب‌سازی حبابی و ضرب ماتریس تعداد دستورهای اضافه شده به ترتیب ۶۵ و ۶۶ دستور است، در حالی که این تعداد در برنامهٔ مرتب‌سازی سریع، ۹۹ دستور می‌باشد، بنابراین افزایش سربار زمان اجرا در برنامهٔ مرتب‌سازی سریع، منطقی به نظر می‌رسد.

جدول ۵- سربار زمان اجرای روش PFC (بر حسب نانو ثانیه)

بارکاری	بارکاری مرجع	بارکاری با اعمال روش	درصد سربار زمان اجرا
مرتب سازی سریع	۴۰۵۸۵۰ ns	۵۴۶۷۹۰ ns	٪۳۴/۷۳
مرتب سازی حبابی	۶۰۲۲۰۰ ns	۷۴۹۲۰۰ ns	٪۲۴/۴۱
ضرب ماتریس	۲۵۴۷۱۰ ns	۳۷۸۹۹۰ ns	٪۴۸/۷۹
درصد میانگین سربار زمان اجرا			٪۳۵/۹۸

۶-۲- نتایج ارزیابی روش MFC

روش محاسبهٔ پارامترها در این بخش مانند بخش ۱-۶ است. در جدول ۶ پارامتر پوشش تصحیح اشکال برای روش MFC محاسبه شده است.

جدول ۶- پوشش تصحیح اشکال در روش MFC

حداکثر تعداد اشکال‌ها	پوشش تصحیح اشکال برحسب مدل اشکال		
	BTMF, BDF	BIF, NBOMF	NBPMF
۱ بیت اشکال	٪۱۰۰	٪۱۰۰	٪۱۰۰
تا۲ بیت لشکال	٪۹۰/۹۱	٪۶۶/۶۷	٪۸۹/۴۶
تا۳ بیت اشکال	٪۷۴/۰۵		٪۷۰/۲۸
تا۴ بیت اشکال	٪۵۳/۰۴		٪۴۷/۳۴
تا۵ بیت اشکال	٪۳۲/۶۷		٪۲۶/۶۵
تا۶ بیت اشکال	٪۱۶/۸۸		٪۱۲/۳۲
تا۷ بیت اشکال	٪۵/۰۹		٪۴/۸۲
تا۸ بیت اشکال	٪۲/۳۱		

مزیت روش MFC بر روش PFC، کشف و تصحیح اشکال دو بیتی در کد عملیاتی دستورها است. معمولاً طول کد عملیاتی در ریزپردازنده‌های مختلف بیش از ۴ بیت است (در ریزپردازندهٔ OpenRISC این کد ۶ بیتی است). لذا کد عملیاتی حداقل در دو میدان ۴ بیتی تقسیم می‌شود و حداقل ۲ بیت اشکال در کد عملیاتی قابل تصحیح است. از سوی دیگر چون MFC برای کل اجزای حافظه (خارجی)، در اینجا ۲۰ کیلو خانه ۳۲ بیتی، پیاده‌سازی می‌شود، قادر است اشکالات چند بیتی رخداده در عملوندهای سایر دستورها را نیز اصلاح کند. اما به دلیل اینکه عملوند دستورها در این ریزپردازندهٔ ۳۲ بیتی حداکثر ۲۶ بیت است (حداقل ۶ بیت برای کدعملیاتی در نظر گرفته می‌شود)، تعدادمیدان‌های ۴ بیتی در عملوند دستورها حداکثر ۷ میدان خواهد بود (و حداکثر ۷ اشکال همزمان قابل تصحیح است) و این مکانیزم قادر به تصحیح ۸ اشکال همزمان در عملوند دستورهای غیرانشعابی نیست.

همانطور که قبلاً هم اشاره شد، روند نزولی پوشش کشف اشکال در اشکالات چند بیتی، به دلیل نزولی بودن احتمال رخداد این اشکال‌ها به کارآیی این روش صدمه‌ای وارد نمی‌کند.

روش MFC تغییرات سخت‌افزاری جدیدی به سیستم شامل ریزپردازنده اعمال کرده است، لذا حجم سخت‌افزار و توان مصرفی ریزپردازنده افزایش یافته است. این مقادیر همان طور که در بخش قبل اشاره شد با کمک ابزار Synopsys بر روی تکنولوژی TSMC 0.18um محاسبه شده است. سربار سخت‌افزار در جدول ۷ ارائه شده است.

جدول ۷- سربار سخت‌افزار روش MFC (برحسب میکرو متر مربع)

	مساحت بر حسب um^2		
سیستم اولیه	سیستم با روش MFC	میزان سربار سخت‌افزار	درصد سربار سخت‌افزار
۱۸۸۲۰۸۰	۲۱۲۹۳۶۰	۲۴۷۲۸۰	٪۱۳/۱۴

در اینجا مانند بخش قبل برای محاسبهٔ سربار توان مصرفی (جدول ۸) از برنامه‌های بارکاری مرتب‌سازی سریع و مرتب‌سازی حبابی با ۳۰ المان و ضرب ماتریس برای دو ماتریس ۴×۴ استفاده شده است. در اینجا نیز توان تلف شده در بار کاری مرتب‌سازی سریع بیش از سایر بارهای کاری است.

جدول ۸- سربار توان مصرفی روش MFC (برحسب میلی وات)

بارکاری	سیستم اولیه	سیستم با روش MFC	درصد سربار توان مصرفی
مرتب‌سازی سریع	۱۵۶/۷۳ mw	۱۸۰/۴۴ mw	٪۱۵/۱۳
مرتب‌سازی حبابی	۱۵۲/۸۹ mw	۱۷۴/۶۹ mw	٪۱۴/۲۶
ضرب ماتریس	۱۵۳/۳ mw	۱۷۵/۷۳ mw	٪۱۴/۶۳
میانگین	۱۵۴/۳۱ mw	۱۷۶/۹۵ mw	٪۱۴/۶۷

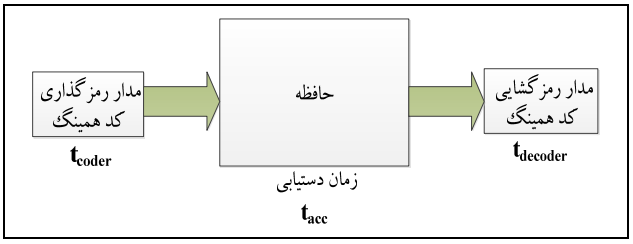
سربار توان مصرفی روش MFC بیشتر از روش PFC است چرا که در این روش محاسبات مربوط به رمزگذاری و رمزگشایی برای تمام دستورها و داده‌ها تکرار خواهند شد و مسلماً توان بیشتری تلف خواهد شد.

در روش MFC برخلاف روش PFC تعداد دستورهای برنامهٔ بارکاری افزایش نمی‌یابد، بلکه طول کلمات حافظه تغییر می‌کند. بنابراین در این روش برای محاسبهٔ سربار حجم حافظه مستقیماً از محاسبهٔ تعداد بایت‌های حافظه استفاده شده است. حجم حافظه خارجی در این آزمایش در اینجا ۲۰ کیلو خانه ۳۲ بیتی (۴ بایتی) است که به ۲۰ کیلو خانه ۵۶ بیتی (۷ بایتی) تبدیل شده است. این مقادیر در جدول ۹ آمده است.

جدول ۹- سربار حجم حافظهٔ روش MFC

حجم حافظه در پردازنده مرجع (کیلو بیت)	۳۲ × ۲۰
حجم حافظه در پردازنده با MFC (کیلو بیت)	۵۶ × ۲۰
درصد سربار حجم حافظه	٪۷۵

در روش MFC با کد همینگ کارایی به دلیل افزایش زمان دستیابی به حافظه، کاهش خواهد یافت. اگر در مدار اولیه زمان دستیابی به حافظه t_{acc} باشد با افزودن مدار رمزگذاری و رمزگشایی به حافظه (شکل ۱۳) زمان دستیابی به حافظه در سیکل نوشتن به $t_{acc} + t_{coder}$ و در سیکل خواندن به $t_{acc} + t_{decoder}$ افزایش خواهد یافت.



شکل ۱۳- افزایش زمان دستیابی به حافظه در روش MFC

مدار رمزگذاری از دو لایه گیت Xor و مدار رمزگشایی از ۴ لایه Xor و یک دیکدر تشکیل شده است. اگر تأخیر انتشار هر گیت Xor را t_{Xor} بنامیم و زمان تأخیر دیکدر (دو لایه AND-OR) را برابر تأخیر انتشار یک گیت Xor فرض کنیم، خواهیم داشت:

زمان دسترسی به حافظه در سیکل نوشتن = ۲ $t_{Xor} + t_{acc}$

زمان دسترسی به حافظه در سیکل خواندن = ۵ $t_{Xor} + t_{acc}$

یکی دیگر از نقاط ضعف این روش، این است که تغییر زمان دستیابی به حافظه در مواردی ممکن است زمان‌بندی سیکل دسترسی به حافظه از طرف پردازنده را دچار مشکل کند.

۷- بحث و مقایسهٔ روش‌ها

در این مقاله، اشکالات SEU در حافظه دستور به عنوان مدل اصلی اشکالات گذرا در ریزپردازنده مد نظر قرار گرفت. برای کشف و تصحیح این اشکال‌ها طبق کارهای پیشین انجام گرفته در این زمینه، دو راه پیشنهاد شده است: راه اول: PFC، پیاده‌سازی سیستم محافظتی در خط لولهٔ ریزپردازنده و راه دوم: MFC، پیاده‌سازی سیستم حفاظتی در حافظه.

روش PFC در خط لوله و روش MFC در حافظهٔ ریزپردازندهٔ OpenRISC به صورت مجزا پیاده‌سازی شدند. اما نکته‌ای که انگیزهٔ ارایهٔ روش MFC بوده، این است که کدهایی مانند همینگ و یا Reed-solomon که برای تحمل‌پذیری در برابر خطاهای یک یا چند بیتی طراحی شده‌اند و بر روی حافظهٔ ریزپردازنده قابل پیاده‌سازی هستند در برابر روش‌های معماری درون پردازنده‌ای مقایسه گردند.

به دلیل ساختارهای متفاوت حافظه و هستهٔ ریزپردازنده تخمین دقیق اثرپذیری هر یک از این دو در برابر اشکال یا خطا، مشکل است. حافظه عنصری است که داده‌ها/دستورها در آن ثبت و نگهداری می‌شوند اما خط لولهٔ ریزپردازنده، مسیر گذر داده است و اطلاعات حالت ماندگاری ندارند. این مورد جدای از تفاوت ذاتی این دو عنصر و تکنولوژی‌های متفاوت به کار رفته در هر یک از این عناصر، عاملی است که باعث اثرپذیرتر شدن حافظه نسبت به اشکال می‌شود.

از سوی دیگر در ریزپردازنده به دلیل سوییچینگ‌های متعدد قابلیت بروز اشکال (برای مثال اشکال cross-talk) افزایش می‌یابد. قرار دادن سیستم کشف و تصحیح اشکال در درون هستهٔ ریزپردازنده نه تنها حافظه بلکه تا حدی مدارهای داخلی ریزپردازنده را در جلوگیری از بروز خطا و کشف و تصحیح اشکالات رخداده کنترل می‌کند چرا که در مرحلهٔ آخر که دستورها از حافظه فراخوانده شده و برای اجرا آماده می‌شوند برای تصحیح دستورها اقدام می‌کند. این سوال مطرح است که پوشش این اشکال‌ها به قیمت تغییر برنامهٔ کاربر، تغییر در هستهٔ ریزپردازنده، توان تلفی این مکانیزم‌ها و کاهش کارآیی ناشی از آن‌ها چقدر ارزش دارد.

برای این مقایسه باید در نظر گرفت که چندین عامل در تفاوت احتمال بروز اشکال در دو سیستم مؤثر است. پارامترهای طراحی یکی از این عوامل تأثیر گذار است. مسلماً از بین دو مدار که سطح ولتاژ مبدأ متفاوتی دارند آن که ولتاژ پایین‌تری دارد در برابر خطا تأثیرپذیرتر است. پارامترهای الکترونیکی یکی دیگر از این عوامل هستند. در دو مدار که از دو تکنولوژی ساخت متفاوت استفاده کرده‌اند، مداری حساس‌تر است که ابعاد ترانزیستورها در آن کوچکتر باشد. مساحت پارامتر بعدی است. اگر ندانیم که داخل دو چیپ چه تکنولوژی به کار رفته و مانند دو جعبهٔ سیاه[۳۶] به آن‌ها نگاه کنیم (و یا حتی از بین دو مدار با تکنولوژی یکسان)، مداری که مساحت بیشتری را به خود اختصاص داده باشد نسبت به اشکال حساس‌تر است.

با فرض این که حافظه و ریزپردازنده روی یک چیپ پیاده‌سازی شوند، تکنولوژی ساخت قطعات و پارامترهای طراحی و الکترونیکی برابری برای مدارها خواهیم داشت. در این حالت مهمترین پارامتری که برای مقایسه باقی خواهد ماند، مساحت است. از آنجایی که در اکثر ریزپردازنده‌ها سطح اختصاص داده شده به حافظه بیشتر از هستهٔ اصلی ریزپردازنده است لذا امکان بروز اشکال در اطلاعات ذخیره شده در حافظه بیشتر است. مقایسه نهایی برای تعیین مزایا و معایب هر یک از این مکانیزم‌ها، در فصل ۶ ارائه شد. لازم به ذکر است که PFC با استفاده از مدل‌های رفتاری کشف و تصحیح اشکال طراحی شده است. لذا در این روش صرفاً با توجه به رفتار مدارها در برابر بروز خطا، مکان‌های دارای احتمال بالاتر بروز اشکال تحت پوشش قرار گرفته است. اما در پیاده‌سازی روش تصحیح اشکال در حافظه، مدل ساختاری کشف و تصحیح اشکال اعمال شده است که در آن برای تمام حافظه (دستور و داده) کد تصحیح اشکال در نظر گرفته شده است.

جدول ۱۰- مقایسه روش ارائه شده با کارهای پیشین

روش	نوع پیاده‌سازی سیستم	مدل خطا	پوشش کشف خطا	پوشش تصحیح خطا	سربار حجم حافظه	سربار زمان اجرا	تأخیر کشف خطا	سربار انرژی مصرفی
PFC	با ریزپردازندهٔ خاص منظوره	MBU	٪۴۶/۸۷	٪۴۶/۸۷	٪۳۱/۶۷	٪۳۵/۹۸	صفر	٪۳/۸۱
MFC	با ریزپردازندهٔ خاص منظوره	MBU	٪۴۶/۸۷ [۲]	٪۴۶/۸۷ [۲]	٪۷۵	صفر	صفر	٪۱۴/۶۷
IMCFEDC [8]	با ریزپردازندهٔ خاص منظوره	SBF	٪۱۰۰ [۱]	٪۸۲	٪۹	٪۸/۶	صفر	٪۱۰
HAP [17]	با ریزپردازندهٔ خاص منظوره	MBU	٪۸۵	صفر	٪۱۱	٪۱/۵	صفر	--
LEON-FT [20]	با ریزپردازندهٔ خاص منظوره	MBU	٪۱۰۰	--	--	--	٪۸	--
ERC32 [21]	با ریزپردازندهٔ خاص منظوره	MBU	--	--	--	--	۵۰ دستور	--
CIC [27]	با استفاده از قابلیت‌های خاص ریزپردازنده‌های عام منظوره	MBU	٪۸۴	صفر	٪۲۸	٪۲۴۵	۸۰۱۹۴ نانو ثانیه	> ٪۲۴۵ [۳]
CFCBTE [11]	با استفاده از قابلیت‌های خاص ریزپردازنده‌های عام منظوره	MBU	٪۵۹	صفر	٪۴۴	٪۳۰۴/۳	۴۱ دستور	٪۳۰۴

[۱] اعلام پوشش کشف خطای ۱۰۰٪ برای روش IMCFEDC (که نزدیکترین روش به مکانیزم BMBC نیز هست) با توجه به این نکته است که در این روش فقط اشکالات حداکثر دو بیتی قابل کشف هستند و صرفاً اشکالات یک‌بیتی قابل تصحیح هستند. لذا این پارامترها برای این روش با توجه به رخداد اشکال MBU بسیار کمتر از این مقدار خواهد بود.

[۲] این مقادیر صرفاً مربوط به میانگین پوشش کشف و تصحیح اشکالات BDF و BTMF در این روش است.

[۳] علامت < به معنای "بیش از" می‌باشد.

از مقایسهٔ دو روش با توجه به نتایج جدول‌های ۱ و ۶ از فصل ۶ بدست می‌آید که پوشش تصحیح اشکال در MFC بالاتر از PFC است (PFC قادر به تصحیح اشکالات دوبیتی در کدعملیاتی دستورها نیست و نیز این روش هیچ نوع اشکالی را در عملوند دستورهای غیر انشعابی کشف و تصحیح نمی‌کند). این نتیجه دور از ذهن نبود، چرا که MFC محافظت در برابر ۸ بیت اشکال همزمان در هر کلمه از حافظه دستور یا حافظهٔ داده را تضمین می‌کند. این در حالی است که PFC صرفاً برای حفاظت دستور اعمال شده است و حفاظت در برابر ۸ بیت اشکال همزمان را فقط در دستورهای انشعاب تضمین می‌کند و در سایر دستورها فقط کد عملیاتی دستور را در برابر یک بیت اشکال محافظت می‌کند.

از نتایج جدول‌های ۲ و ۷ بر می‌آید که سر بار سخت‌افزار در روشMFC بیشتر از روش PFC است (۱۳/۱۴٪ در برابر ۱۱/۰٪). در روش MFC حجم حافظهٔ افزوده شده سبب بالاتر بودن حجم سخت‌افزار مصرفی شده است. اما بدون در نظر گرفتن سربار سخت‌افزار حاصل از افزونگی داده در این روش، تغییرات سخت‌افزاری در روش PFC بیشتر بوده است. این مدارها با اینکه حجم بیشتری دارند ولی به میزان سخت‌افزار اضافه شده در روش MFC توان تلف نمی‌کنند (۱۴/۶۷٪). چرا که در روش MFC، این مدارها برای هر فراخوانی دستور یا واکشی داده، عملیات رمزگشایی را انجام می‌دهند و برای هر بار ذخیرهٔ داده، عملیات رمزگذاری را انجام می‌دهند و در نتیجه توان بالایی مصرف می‌کنند.

سربار حجم حافظه در روش MFC بسیار بیشتر از روش PFC است (۷۵٪ در برابر ۳۱/۶۷٪). در روش PFC با توجه به رفتار سیستم‌ها هنگام بروز خطا، برای تعداد خاصی دستور مکانیزم‌های بیان شده پیاده‌سازی شده‌اند. در واقع این پارامتر نقطهٔ قوت روش PFC است که با سربار حجم حافظهٔ کمتری توانسته است پوشش کشف اشکال مناسبی ارائه بدهد.

روش MFC نیازی به اعمال تغییرات نرم‌افزاری در برنامهٔ بارکاری ندارد. این خصوصیت می‌تواند یک مزیت برای روش MFC محسوب شود. البته توان مصرفی بالای این روش (۱۴/۶۷٪) نکتهٔ بسیار مهمی است که در کاربردهایی چون سیستم‌های Low-power بایستی مورد توجه قرار بگیرد.

نهایتاً انتخاب هر یک از این روش‌ها با توجه به امکانات موجود و پارامترهای مورد نیاز صورت می‌گیرد. برای ایجاد امکان مقایسه بین روش این مقاله با روش‌های ارائه شدهٔ پیشین، در اینجا با توجه به مقادیری که توسط خود مقالات ارائه شده است به مقایسهٔ بعضی از این پارامترها پرداخته شده است. پارامترهای ارائه شده (جدول ۱۰) برای روش‌های پیاده‌سازی شده در این مقاله، همگی از میانگین مقادیر جدول‌های فصل ۶ بدست آمده‌اند.

از مقایسهٔ نتایج ارائه شده در جدول ۱۰ مشخص است که یکی از مزایای روش‌های ارائه شده در این مقاله، بلادرنگ بودن کشف اشکالات بیان شده در مدل اشکال ارائه شده برای این روش‌ها است. در واقع هر دوی این روش‌ها با سربار مصرف انرژی کمتر و نیز سربار زمان اجرای مناسبی، پوشش کشف اشکال قابل توجهی را ارائه می‌دهند.

۸- نتیجه‌گیری

در این مقاله دو روش مستقل، برای کشف و تصحیح اشکال SEU، در حافظه و خط لولهٔ ریزپردازندهٔ OpenRISC پیاده‌سازی شد. اولین روش که PFC نامیده می‌شود، به پیاده‌سازی دو مکانیزم BMBC و OSBC در خط لولهٔ ریزپردازنده می‌پردازد. مکانیزم BMBC حداکثر هشت بیت اشکال همزمان در دستور انشعاب را کشف و تصحیح می‌کند. مکانیزم OSBC کشف و تصحیح یک بیت اشکال در کد عملیاتی هر دستور را تضمین می‌کند. روش دوم که MFC نام دارد، به پیاده‌سازی طرحی در حافظهٔ ریزپردازنده می‌پردازد. این روش ۸ بیت اشکال همزمان در هر دستور را کشف و تصحیح می‌نماید. نتایج حاصل از ارزیابی دو روش نشان می‌دهد که MFC پوشش تصحیح اشکال بهتری ارائه می‌دهد ولی افزونگی داده و در نتیجه سربار سخت‌افزاری در این روش بیشتر است (۱۳/۱۴٪). این در حالی است که روش PFC با داشتن پوشش تصحیح اشکال مناسب، سربار حجم برنامهٔ کمتری و سربار حجم سخت‌افزار مصرفی کمتری به ریزپردازنده تحمیل می‌کند (۱۱/۱۰٪) و سربار توان مصرفی کمتری نسبت به روش MFC دارد (۳/۸۱٪ در برابر ۱۴/۶۷٪). انتخاب هر یک از این روش‌ها بنابر نیازهای سیستم‌های مختلف صورت می‌گیرد.

مراجع

[1] G. Neuberger, and F. De Lima, L. Carro, R. Reis, "A Multiple Bit Upset Tolerant SRAM Memory," *ACM Trans.*

[15] A. Rajabzadeh, and S. G.Miremadi, "CFCET: A hardware-based control flow checking technique in COTS processors using execution tracing," *Journal of Microelectronics Reliability*, pp. 959-972, May-June 2006.

[16] N. Farazmand, *System-Level Monitoring of an Embedded Processor Using Bilt-in Components*, M.Sc. Thesis (in Persian), Department of Computer Engineering, Sharif University of Technology, 2008.

[17] R. G. Ragel, and S. Parameswaran, "Hardware Assisted Pre-Emptive Control Flow Checking for Embedded Processors to Improve Reliability," *Proc. of the 4th international conference on Hardware/software codesign and system synthesis*, pp. 100-105, October 2006.

[18] R. G. Ragel, and SRI. Parameswaran, "A Hybrid Hardware Software Technique to improve Reliability in Embedded Processors," *Proc. of ACM Transactions on Computational Logic*, pp. 100–117, 2009.

[19] J. YAO, R. Watanabe, T. Nakada, H. Shimada, Y. Nakashima, and K. Kobayashi, "A Minimal Roll-Back Recovery Scheme for Fault Toleration in Pipeline Processors," *Proc. of the Pacific Rim International Symposium on Dependable Computing*, pp. 237-238, 2010.

[20] J. Gaisler, "A Portable and Fault-Tolerant Microprocessor Based on the SPARC V8 Architecture, European Space Agency," Noorwijk, *Proc. of the International Conference on Dependable Systems and Networks*, DSN, pp. 409 - 415, 2002.

[21] T. Michel, R. Leveugle, F. Gaume, and R. Roane,"An Application Specific Microprocessor with Two-Level Built-In Control Flow Checking Capabilities," *Proc. of the Euro ASIC'92*, pp. 310-313, June 1992.

[22] A. Rajabzadeh, and S. G. Miremadi, "A Hardware Approach to Concurrent ErrorDetection Capability Enhancement in COTS Processors," *Proc. of 11th Pacific RimInternational Symposium on Dependable Computing*, December 2005.

[23] H. R. Zarandi, M. Maghsoudloo, and N. Khoshavi, "Two Efficient Software Techniques to Detect and Correct Control-flow Errors," *Proc. of the Pacific Rim International Symposium on Dependable Computing*, pp. 141-148, 2010.

[24] P. Bernardi, L. Bolzani, M. Rebaudengo, M. Sonza Reorda, F. Vargas, and M. Violante, "A New Hybrid Fault Detection Technique for Systems-on-a-Chip," *Proc. of the IEEE TRANSACTION ON COMPUTERS*, pp. 185-198, February 2006.

[25] N. Oh, P. P. Shirvani, and E. J. McCluskey, "Control-Flow Checking by Software Signatures," *IEEE Trans. on Reliability*, pp. 111-122, March 2002.

on Design Automation of Electronic Systems, pp. 577-590, October 2003.

[2] N. Seifert, et al., "Frequency Dependence of Soft Error Rates for Sub-Micron CMOS Technologies," *Proc. of International Electron Devices Meeting, Technical Digest IEDM*, pp. 323-326, 2001.

[3] P. Marwedel, *Embedded System Design*, Springer, Netherland, 2006.

[4] E. Touloupis, and V. A. Chouliaras, "Study of the Effects of SEU-Induced Faults on a Pipeline-Protected Microprocessor," *IEEE Transaction on Computer*, vol. 56, no. 12, pp. 1585-1596, December 2007.

[5] F. X. Ruckerbauer, and G. Georgakos, "Soft Error Rates in 65nm SRAMs—Analysis of New Phenomena," *On-Line Testing Symposium, IOLTS 07.13th IEEE International*, pp. 203-204, July 2007.

[6] A. Mahmood, and E. J. McCluskey, "Concurrent Error Detection Using Watchdog Processors – A Survey," *IEEE Trans. on Computers*, pp. 160-174, February 1988.

[7] J. Gaisler, "LEON-1 Processor – First Evaluation Results," *European Space Components conference (ESCCON). Proc. of the European Space Components Conferences ESCCON, ESA-SP*, vol. 439, pp. 183-188, 2000.

[8] N. Farazmand, M. Fazeli, and S. G. Miremadi, "FEDC: Control Flow Error Detection and Correction for embedded systems without program interruption," *Proc. of Third International Conference on Availability, Reliability and Security*, pp. 33-38, March 2008.

[9] http://www.opencores.com (2010/9/20).

[10] S. G. Miremadi, J. Ohlsson, M. Rimen, and J. Karlsson, "Use of Time, Location and Indtruction Signature for Control Flow Checking," *Proc. of the DCCA-6 International Conference*, IEEE computer Society Press, 1988.

[11] B. Fazeli, *Design and Implementation of a Fault-Tolerant Code Generator Tool for MPC555-based Embedded systems*, M.Sc. Thesis (in Persian), Department of Computer Engineering, Sharif University of Technology, 2005.

[12] M. Namjoo, "Design of Concurrently Test Able Micro Programmed Control Units," *Proc. of the 15th annual workshop on Microprogramming*, 1982.

[13] A. Li, and B. Hong, "On-line control flow error detection using relationship signatures among basic blocks," *Proc. of Computers and Electrical Engineering*, pp. 132-141, January 2010.

[14] E. Borin, C. Wang, Y. Wu, and G. Araujo,"Software based transparent and comprehensive control-flow error detection," *Proc. of International Symposium on Code Generation and Optimization*, pp. 333-345, March 2006.

شیلان پارساییان با دریافت مدرک کارشناسی مهندسی برق (گرایش الکترونیک) در سال ۱۳۸۳ از دانشگاه رازی (کرمانشاه) فارغ التحصیل شد. در فاصلهٔ سال‌های ۱۳۸۴ تا ۱۳۸۹ در شرکت پتروشیمی کرمانشاه به عنوان مهندس ابزار دقیق مشغول به کار بود. او در سال ۱۳۹۰ مدرک کارشناسی ارشد خود را از دانشگاه رازی در رشتهٔ مهندسی برق (گرایش الکترونیک) اخذ نمود و در حال حاضر نیز مشغول تحصیل در رشتهٔ مهندسی برق در مقطع دکتری در دانشگاه صنعتی چالمرز (سوئد) می‌باشد. حوزه‌های تحقیقی مورد علاقهٔ ایشان سیستم‌های تعبیه شده و سیستم‌های کنترل خطی نظارت بر عملکرد می‌باشد.
آدرس پست‌الکترونیکی ایشان عبارت است از:
shilan@student.chalmers.se

امیر رجب‌زاده فارالتحصیل دورهٔ دکتری مهندسی کامپیوتر (معماری کامپیوتر) از دانشگاه صنعتی شریف در سال ۱۳۸۳ است. همچنین ایشان دورهٔ کارشناسی ارشد و کارشناسی را به ترتیب در رشته مهندسی کامپیوتر (معماری کامپیوتر) از دانشگاه صنعتی شریف و مهندسی برق (مخابرات) از دانشگاه تهران گذرانده است. از سال ۱۳۸۳ تاکنون استادیار دانشگاه رازی بوده و در سال‌های ۱۳۸۶ تا ۱۳۸۸ دارای مسئولیت معاونت دانشکده فنی مهندسی بوده‌اند. حوزهٔ فعالیت علمی ایشان معماری کامپیوتر، معماری اتکاپذیر، سیستم‌های تعبیه شده و سخت افزار با قابلیت پیکربندی مجدد می‌باشد.
آدرس پست‌الکترونیکی ایشان عبارت است از:
rajabzadeh@razi.ac.ir

اطلاعات بررسی مقاله:

تاریخ ارسال: ۹۰/۲/۱۰
تاریخ اصلاح: ۹۱/۱/۱۵
تاریخ قبول شدن: ۹۱/۱/۲۰
نویسنده مرتبط: دکتر امیر رجب‌زاده، دانشکده فنی و مهندسی، دانشگاه رازی، کرمانشاه، ایران.

[1] Faults
[2] Embedded Systems
[3] Single Event Upset (SEU)
[4] Single Bit Flip (SBF)
[5] Multi-Bit Upset (MBU)
[6] Circuit Level Techniques
[7] System Level Techniques
[8] Internal Circuit Level
[9] External Circuit Level
[10] Branch Multi-Bit Correction (BMBC)
[11] Opcode Single-Bit Correction (OSBC)
[12] Instruction Memory
[13] Data Memory
[14] Concurrent Error Detection
[15] ASIC (Application Specific Integration Circuit)
[16] Triple Module Redundancy
[17] Cyclic Redundancy Code (CRC)
[18] Check Point
[19] Flag
[20] Performance Test
[21] Debugging
[22] Branch Trace Exception (BTE)
[23] Performance Monitoring Counter (PM-Counter)
[24] Execution Tracing
[25] COTS-Based Design
[26] Hybrid
[27] Watchdog
[28] Branch Insertion Fault (BIF)
[29] Branch Target Modification Fault (BTMF)
[30] Branch Deletion Fault (BDF)
[31] Non-Branch Opcode Modification Fault (OMF)
[32] Non-Branch Oprand Modification Fault (NBPMF)
[33] Post Processor
[34] Low-Power
[35] Fault Tolerant
[36] Black Box