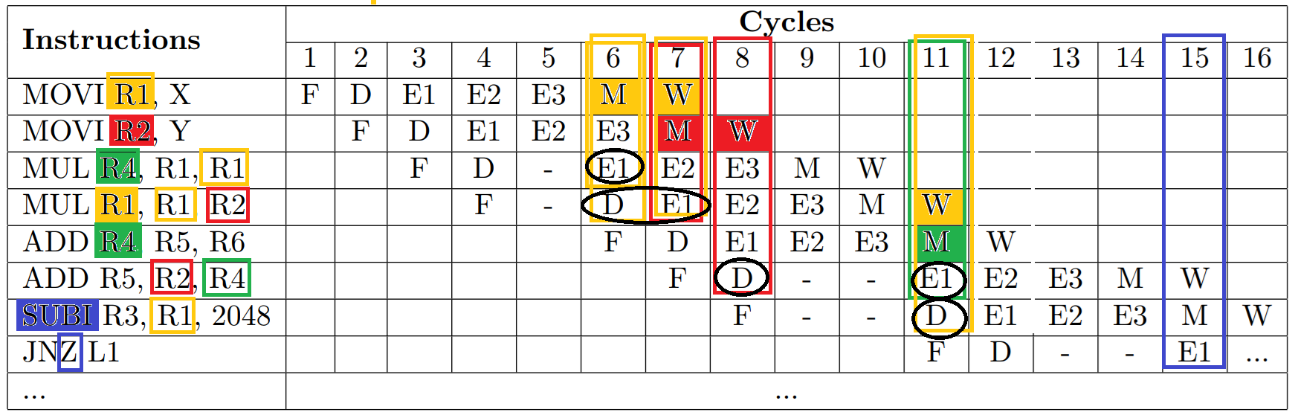
**1.**

**آ)** من توی شکل Data های مشترک و زمان ساخت و استفادشون رو علامت زدم (خونه هایی که توشون پر شده داده های درست رو دارن و خونه هایی که دورشون دایره مشکی داره داده رو نیاز دارن):

**

*همانطور که در تصویر مشاهده میکنید، 4 کلاک مستعد Forwarding داده هستند. در دستوراتی که از Data رجیستر ها استفاده میکنند، یا باید داده در مرحله Decode که داده ها از رجیستر خوانده میشوند، به ما برسند( یا Write back قبلا انجام شده باشه یا داده پاس داده شود) و یا باید در کلاک بعد به E1 پاس داده شوند. پس اگر فرض کنیم که Data Forwarding بین قسمت های مرحله فعلی که داده را دارند انجام میشود، داریم:*

* *میان E1, M Data Forwardingوجود دارد چون در کلاک 6 ام به R1 ، در کلاک 7 ام به R2 و در کلاک 11 ام به R4 نیاز داریم.*
* *میان E1, W و یاD,M Data Forwardingوجود دارد چون در دستور 4 ام به R1 نیاز داریم و این نیاز را یا در کلاک 7 ام و یا در کلاک 6 ام باید هندل کنیم.*
* *میان D,W Data Forwardingوجود دارد (اگر latch داشته باشیم نیازی نیست) چون در کلاک 8 ام به R2 و در کلاک 11 ام به R1 نیاز داریم.*

*پ.ن: در دستور آخر پرش به پرچم z نیاز دارد ولی من معتقدم که این فلگ در مرحله E3 در کلاک 14 ام ست شده است و نیازی به چک کردن دستی R13 و Data Forwarding برای آن نیست...*

**ب)** *این پردازنده از interlocking به صورت سخت افزاری استفاده میکند زیرا Hazard هارا با stall و Data Forwarding (و حتی شاید با latch) حل کرده است (اگر با لحاظ کردن تغییراتی در کد مانند استفاده نکردن دستورات وابسته، استفاده از no-op و یا code reordering آن هارا هندل کرده بود، نرم افزاری حساب میشد)*

**ج)** *چون X=4 است پس در ابتدای این کد R1 برابر با 4 شده است و سپس وارد حلقه ای شده است که با ضرب R2 ( که چون Y=4 بوده و در جای دیگر تغییری نکرده است، درین کد برابر با 2 است) در آن، آن را تغییر میدهد و با رسیدن آن یه 2048 از حلقه خارج میشود. برای اینکه R1= شود باید 8 بار MUL R1, R1, R2 اجرا شود. همچنین از آنجا که در حال واکشی MUL R4, R1, R1 هستیم، پس وارد دور بعدی حلقه و اجرای 9 ام آن شدیم. اولین واکشی دستور گفته شده در کلاک 3ام است و دومین واکشی آن در کلاک 12 ام است. اما در اجرای دوم حلقه بخاطر استفاده دستور JNZ از ALU ، 2 کلاک کند تر خواهد شد و از طرفی چون دو دستور اول برنامه قبلا اجرا شده اند، stall کلاک ششم اجرا نخواهد شد، پس ازین به بعد 10 کلاک یکبار انجام میشود. پس:82 T=12 + 7\*10=*

**د)** *2 دستور قبل از حلقه، 8 بار اجرای کامل حلقه (6 دستور در هر حلقه) و یک بار دستور اول حلقه : N=2+8\*6+1=51*

**ه)** *برای خارج شدن از حلقه باید R1 برابر 2048 شود که یعنی از زمانی که R1=1024 است، تنها یک بار دیگر باید حلقه انجام شود. پس در چرخه T+10 دستور آخر واکشی میشود و برای اجرای کامل آن بخاطر 2 تا stall دستور قبل و وابسته نبودن داده خودش به 2+6=8 کلاک دیگر نیاز داریم. پس کلاک 82+10+8 = 100 ام آخرین چرخه برنامه است.*

**2.** *میدانیمthroughput به تعداد دستورات یا عملیات‌هایی اطلاق می‌شود که پردازنده قادر است در یک واحد زمان انجام دهد. پس برای بیشینه شدن آن کافیست زمان هرکلاک را کمینه کنیم:*

**3.** *بدون دور زدن:*

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *12* | *11* | *10* | *9* | *8* | *7* | *6* | *5* | *4* | *3* | *2* | *1* | *Instruction* |
|  |  |  |  |  |  |  | *W* | *M* | *X* | *D* | *F* | *Sub $2, $3, $1* |
|  |  |  |  | *W* | *M* | *X* | *D* | *d\** | *d\** | *F* |  | *Lw $5, 0($2)* |
|  | *W* | *M* | *X* | *D* | *d\** | *d\** | *F* |  |  |  |  | *Addi $4, $5, 1* |
| *W* | *M* | *X* | *D* | *F* |  |  |  |  |  |  |  | *Add $5, $3, $1* |

*دور زدن کامل:*

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *12* | *11* | *10* | *9* | *8* | *7* | *6* | *5* | *4* | *3* | *2* | *1* | *Instruction* |
|  |  |  |  |  |  |  | *W* | *M* | *X* | *D* | *F* | *Sub $2, $3, $1* |
|  |  |  |  |  |  | *W* | *M* | *X* | *D* | *F* |  | *Lw $5, 0($2)* |
|  |  |  |  | *W* | *M* | *X* | *d\** | *D* | *F* |  |  | *Addi $4, $5, 1* |
|  |  |  | *W* | *M* | *X* | *D* | *d\** | *F* |  |  |  | *Add $5, $3, $1* |

**4. آ)** *کد a :*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *14* | *13* | *12* | *11* | *10* | *9* | *8* | *7* | *6* | *5* | *4* | *3* | *2* | *1* | *Instruction* |
|  |  |  |  |  |  |  |  |  | *W* | *M* | *X* | *D* | *F* | *LW R2, 0(R2)* |
|  |  |  |  |  |  |  | *W* | *M* | *X* | *-* | *D* | *F* |  | *BEQ R2, R0, Label1* |
|  |  |  |  |  |  | *W* | *M* | *X* | *D* | *-* | *F* |  |  | *LW R2, 0(R2)* |
|  |  |  |  | *W* | *M* | *X* | *-* | *D* | *F* |  |  |  |  | *BEQ R2, R0, Label1* |
|  | *W* | *M* | *X* | *D* | *F* |  |  |  |  |  |  |  |  | *OR R2, R2, R3* |
| *W* | *M* | *X* | *D* | *F* |  |  |  |  |  |  |  |  |  | *SW R2, 0(R5)* |

*کد b :*

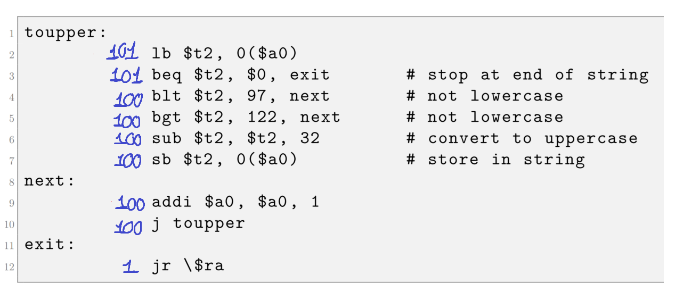
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *14* | *13* | *12* | *11* | *10* | *9* | *8* | *7* | *6* | *5* | *4* | *3* | *2* | *1* | *Instruction* |
|  |  |  |  |  |  |  |  |  | *W* | *M* | *X* | *D* | *F* | *LW R1, 0(R1)* |
|  |  |  |  |  |  |  | *W* | *M* | *X* | *-* | *D* | *F* |  | *BEQ R2, R0, Label2* |
|  |  |  |  | *W* | *M* | *X* | *D* | *F* |  |  |  |  |  | *LW R3, 0(R2)* |
|  |  | *W* | *M* | *X* | *-* | *D* | *F* |  |  |  |  |  |  | *BEQ R3, R0, Label1* |
|  | *W* | *M* | *X* | *D* | *-* | *F* |  |  |  |  |  |  |  | *BEQ R2, R0, Label2* |
| *W* | *M* | *X* | *D* | *F* |  |  |  |  |  |  |  |  |  | *SW R1, 0(R2)* |

**ب)** *کد a :*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *14* | *13* | *12* | *11* | *10* | *9* | *8* | *7* | *6* | *5* | *4* | *3* | *2* | *1* | *Instruction* |
|  |  |  |  |  |  |  |  |  | *W* | *M* | *X* | *D* | *F* | *LW R2, 0(R2)* |
|  |  |  |  |  |  |  | *W* | *M* | *X* | *-* | *D* | *F* |  | *BEQ R2, R0, Label1 (delay slot)* |
|  |  |  |  |  |  | *W* | *M* | *X* | *D* | *-* | *F* |  |  | *OR R2, R2, R3* |
|  |  |  |  |  | *W* | *M* | *X* | *D* | *F* |  |  |  |  | *LW R2, 0(R2)* |
|  |  |  | *W* | *M* | *X* | *-* | *D* | *F* |  |  |  |  |  | *BEQ R2, R0, Label1* |
|  |  | *W* | *M* | *X* | *D* | *-* | *F* |  |  |  |  |  |  | *OR R2, R2, R3 (delay slot)* |
| *W* | *M* | *X* | *D* | *F* |  |  |  |  |  |  |  |  |  | *SW R2, 0(R5)* |

*کد b :*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *14* | *13* | *12* | *11* | *10* | *9* | *8* | *7* | *6* | *5* | *4* | *3* | *2* | *1* | *Instruction* |
|  |  |  |  |  |  |  |  |  | *W* | *M* | *X* | *D* | *F* | *LW R1, 0(R1)* |
|  |  |  |  |  |  |  | *W* | *M* | *X* | *-* | *D* | *F* |  | *BEQ R2, R0, Label2* |
|  |  |  |  |  |  | *W* | *M* | *X* | *D* | *-* | *F* |  |  | *LW R3, 0(R2)*  *(delay slot)* |
|  |  |  |  | *W* | *M* | *X* | *D* | *F* |  |  |  |  |  | *BEQ R3, R0, Label1* |
|  |  |  | *W* | *M* | *X* | *D* | *F* |  |  |  |  |  |  | *ADD R1, R3,R1* |
|  |  | *W* | *M* | *X* | *D* | *F* |  |  |  |  |  |  |  | *BEQ R2, R0, Label2* |
|  | *W* | *M* | *X* | *D* | *F* |  |  |  |  |  |  |  |  | *LW R3, 0(R2)*  *(delay slot)* |
| *W* | *M* | *X* | *D* | *F* |  |  |  |  |  |  |  |  |  | *SW R1, 0(R2)* |

**

**5. آ)** *در تصویر تعداد اجرای هر دستور را نوشتم. پس سرجمع 803 دستور اجرا میشود.*

**ب)** *در معماری single cyvle همواره CPI برابر 1 است.*

**ج)** *دستور دوم هر بار 2 تا stall میخورد چون داده t2 در پایان مرحله Memory تازه میرسد و در کلاک بعد قبل از آنکهWrite back شود میتوانیم آن را با DataForwarding به ID برسانیم. در کل 3 عدد دستور پرش شرطی داریم که سر جمع 301 بار اجرا میشوند، بنابراین در آن 10% مواقع که اشتباه پیشبینی میکنند به طور تقریبی 30.1 کلاک حروم میکنند. پس داریم:*

**د)** *خط لوله ای نسبت به تک چرخه ای 209% بهبود دارد.*

**6. آ)** *مکانیزم پیش‌بینی‌کننده‌ی ۲ بیتی(2-bit saturating counter) یکی از روش‌های پیش‌بینی پرش (branch prediction) در معماری کامپیوتر است. برای هر دستور پرش شرطی، یک شمارنده‌ی ۲ بیتی نگه داشته می‌شود. این شمارنده می‌تواند ۴ مقدار داشته باشد:*

|  |  |  |
| --- | --- | --- |
| *مقدار شمارنده* | *حالت* | *پیش‌بینی* |
| *00* | *Strongly Not Takenخیلی احتمال عدم پرش* | *Not Taken* |
| *01* | *Weakly Not Taken احتمال کم عدم پرش* | *Not Taken* |
| *10* | *Weakly Taken احتمال کم پرش* | *Taken* |
| *11* | *Strongly Takenخیلی احتمال پرش* | *Taken* |

*عملکرد:*

1. *در ابتدا برای هر پرش، شمارنده در یک مقدار تنظیم می‌شود (مثلاً 11 = Strongly Taken)*
2. *در هر بار اجرای پرش:*

* *اگر پیش‌بینی درست باشد → شمارنده تغییر نمی‌کند یا به سمت همان جهت حرکت می‌کند.*
* *اگر پیش‌بینی غلط باشد → شمارنده به سمت جهت مخالف یکی کم یا زیاد می‌شود.*

1. *به این ترتیب، سیستم در مقابل تغییرات ناگهانی در رفتار پرش‌ها مقاوم است.*

*یک پیش‌بینی‌کننده ۱ بیتی با کوچکترین تغییر رفتار سریعاً پیش‌بینی را عوض می‌کند اما ۲ بیتی با دو مرحله‌ی میانی (weakly taken / weakly not taken) پایداری بیشتری دارد.*

*مزایا:*

* *پایداری بیشتر از پیش‌بینی ۱ بیتی*
* *مقاوم در برابر نوسان‌های کوچک در رفتار پرش*
* *بسیار ساده برای پیاده‌سازی در سخت‌افزار (فقط یک شمارنده ۲ بیتی برای هر آدرس پرش)*