به نام خدا

دانشکده مهندسی کامپیوتر

درس معماری کامپیوتر
نیم‌سال دوم ۰۴-۰۳
استاد: دکتر اسدی

پاسخنامه تمرین سری اول

پاسخنامه تمارین تئوری

۱. (آ) مقدار CPI هر کامپیوتر به شرح زیر است:

$$CPI_{C1} = (0.16 \times 6) + (0.1 \times 8) + (0.08 \times 10) + (0.66 \times 3) = 4.54$$
$$CPI_{C2} = (0.16 \times 20) + (0.1 \times 32) + (0.08 \times 66) + (0.66 \times 3) = 13.66$$

(ب) مقدار MIPS هر پردازنده به شرح زیر است:

$$MIPS = \frac{f}{CPI \times 10^6}$$

$$MIPS_{C1} = \frac{400 \times 10^6}{4.54 \times 10^6} = 88.1$$
$$MIPS_{C2} = \frac{400 \times 10^6}{13.66 \times 10^6} = 29.28$$

(ج) زمان اجرای برنامه در هر پردازنده به شرح زیر است:

$$CPU_{TIME} = \frac{numberofinstruction}{MIPS \times 10^6}$$

$$CPU_{C1} = \frac{12000}{88.1 \times 10^6} = 136.2\mu s$$
$$CPU_{C2} = \frac{12000}{29.28 \times 10^6} = 410\mu s$$

(د) پس از تسریع، زمان اجرای دستورات به صورت زیر می‌شود:

$$CPI_{Improved} = 0.16 \times 10 + 0.1 \times 8 + 0.08 \times 22 + 0.66 \times 3 = 6.14$$

$$SpeedUp = \frac{13.66}{6.14} = 2.225$$

نوع دستور	درصد استفاده دستورات در برنامه	زمان اجرای دستورات در C۲ (تعداد ساعت)
جمع ممیز شناور	۱۶%	۱۰
ضرب ممیز شناور	۱۰%	۸
تقسیم ممیز شناور	۸%	۲۲
سایر دستورات	۶۶%	۳

۲. (آ) طبق فرمول زیر داریم:

$$ExecutionTime = InstructionCount \times CPI \times ClockCycleTime$$

بنابراین داریم:

$$CPI = \frac{ExecutionTime}{InstructionCount \times ClockCycleTime}$$

برای برنامه داریم:

$$CPI_A = \frac{1}{1 \times 10^9 \times 10^{-9}} = 1$$

$$CPI_B = \frac{1.4}{1.2 \times 10^9 \times 10^{-9}} = 1.167$$

(ب) طبق فرمول زیر داریم:

$$ExecutionTime = \frac{InstructionCount \times CPI}{ClockRate}$$

بنابراین:

$$\frac{ClockRate_A}{ClockRate_B} = \frac{InstructionCount_A \times CPI_A}{InstructionCount_B \times CPI_B}$$

$$\frac{ClockRate_A}{ClockRate_B} = \frac{1 \times 10^9 \times 1}{1.2 \times 10^9 \times 1.167} \approx 0.714$$

(ج) طبق فرمول زیر داریم:

$$SpeedUp = \frac{ExecutionTime_{Old}}{ExecutionTime_{New}}$$

بنابراین داریم:

$$ExecutionTime_{New} = 600 \times 10^6 \times 1.1 \times 10^{-9}$$

بنابراین برای برنامه داریم:

$$SpeedUp_A = \frac{ExecutionTime_A}{ExecutionTime_{New}} = \frac{1}{0.66} \approx 1.52$$

$$SpeedUp_B = \frac{ExecutionTime_B}{ExecutionTime_{New}} = \frac{1.4}{0.66} \approx 2.12$$

۳. (آ) **تعریف معماری‌های** SIMD **و** MIMD

SIMD یک مدل پردازشی است که در آن یک واحد کنترل، یک دستورالعمل را هم‌زمان بر روی چندین داده اعمال می‌کند. این معماری برای عملیات‌هایی که نیاز به پردازش موازی روی مجموعه‌ای از داده‌های یکسان دارند، مناسب است.

MIMD در این معماری، چندین پردازنده به‌طور مستقل کار کرده و می‌توانند دستورالعمل‌های مختلفی را روی داده‌های مختلف اجرا کنند. هر پردازنده واحد کنترل مخصوص به خود را دارد که امکان اجرای مستقل وظایف را فراهم می‌کند.

تفاوت‌های SIMD **و** MIMD

- در SIMD، تمام پردازنده‌ها یک دستورالعمل را روی داده‌های مختلف اجرا می‌کنند، در حالی که در MIMD هر پردازنده می‌تواند دستورالعمل متفاوتی را اجرا کند.
- SIMD ساده‌تر و کارآمدتر برای پردازش‌های برداری و محاسبات هم‌زمان است، در حالی که MIMD انعطاف‌پذیری بیشتری برای اجرای وظایف متنوع دارد.
- SIMD معمولاً در پردازنده‌های گرافیکی و پردازش‌های چندرسانه‌ای کاربرد دارد، در حالی که MIMD در پردازنده‌های چندهسته‌ای و سیستم‌های توزیع‌شده استفاده می‌شود.

مثال‌ها در کامپیوترهای شخصی

۱. SIMD: پردازنده‌های گرافیکی (GPU) که در پردازش تصاویر و اجرای الگوریتم‌های هوش مصنوعی کاربرد دارند.

۲. MIMD: پردازنده‌های چندهسته‌ای (Multi-Core CPU) که اجرای هم‌زمان چندین برنامه را ممکن می‌کنند.

برتری نسبتی این دو معماری نسبت به یکدیگر

- SIMD برای پردازش‌های برداری، محاسبات گرافیکی، و پردازش داده‌های موازی بسیار مناسب است.
- MIMD در پردازش‌های عمومی، سیستم‌های چندوظیفه‌ای، و اجرای وظایف پیچیده که نیاز به پردازش مستقل دارند، بهتر عمل می‌کند.

(ب) **تعریف** Out-of-Order Execution

Out-of-Order Execution (یا به اختصار OoOE) یک تکنیک پردازشی در پردازنده‌های مدرن است که به جای اجرای دستورالعمل‌ها به ترتیب ورود آن‌ها (In-Order Execution)، پردازنده می‌تواند دستورالعمل‌های آماده اجرا را زودتر از دستورالعمل‌هایی که منتظر داده یا نتیجه محاسباتی هستند، اجرا کند. این تکنیک باعث افزایش استفاده از واحدهای محاسباتی و کاهش تأخیر ناشی از وابستگی‌های داده‌ای می‌شود.

تاثیر Out-of-Order Execution **بر عملکرد پردازنده**

- افزایش کارایی پردازنده: با اجرای دستورالعمل‌های آماده، پردازنده می‌تواند از تمام واحدهای محاسباتی خود به‌صورت بهینه استفاده کند.
- کاهش تأخیر ناشی از وابستگی داده‌ها: اگر یک دستورالعمل منتظر نتیجه یک محاسبه باشد، پردازنده می‌تواند به جای توقف، سایر دستورالعمل‌های مستقل را اجرا کند.
- بهبود Instruction Throughput: این تکنیک منجر به اجرای هم‌زمان تعداد بیشتری دستورالعمل در هر چرخه پردازنده می‌شود.

مراحل اجرای Out-of-Order Execution

۱. دریافت و رمزگشایی دستورالعمل‌ها (Instruction Fetch Decode): پردازنده دستورالعمل‌ها را از حافظه دریافت می‌کند.
۲. تحلیل وابستگی داده‌ای (Dependency Analysis): بررسی می‌شود که کدام دستورالعمل‌ها برای اجرا آماده‌اند.
۳. زمان‌بندی و تخصیص واحدهای اجرایی (Instruction Scheduling): دستورالعمل‌های مستقل که آماده اجرا هستند به واحدهای پردازشی تخصیص داده می‌شوند.
۴. اجرا (Execution): دستورالعمل‌ها در ترتیب بهینه اجرا می‌شوند.
۵. بازچینی مجدد (Reorder Buffer): نتایج به ترتیب اصلی بازگردانده می‌شوند تا از حفظ ترتیب برنامه اطمینان حاصل شود.

چالش‌ها و محدودیت‌های Out-of-Order Execution

- پیچیدگی سخت‌افزاری: نیاز به واحدهای اضافی مانند Reorder Buffer، Reservation Stations و مکانیزم‌های بررسی وابستگی داده‌ای دارد.
- مصرف انرژی بیشتر: به دلیل مدیریت پیچیده‌تر دستورالعمل‌ها، مصرف انرژی افزایش می‌یابد.
- محدودیت‌های وابستگی حافظه‌ای: وابستگی‌های خواندن و نوشتن داده‌ها در حافظه باید مدیریت شوند تا از اجرای نادرست جلوگیری شود.

نتیجه‌گیری

Out-of-Order Execution یکی از مهم‌ترین پیشرفت‌ها در طراحی پردازنده‌های مدرن است که باعث افزایش کارایی از طریق اجرای موازی و کاهش تأخیرهای غیرضروری می‌شود. این تکنیک در کنار روش‌هایی مانند Branch Prediction و Speculative Execution، عملکرد پردازنده‌های امروزی را به‌طور چشم‌گیری بهبود بخشیده است.

۴. (آ) CPI اصلی

$$(3 \text{ x } 0.35) + (5 \text{ x } 0.20) + (4 \text{ x } 0.15) + (2 \text{ x } 0.25) + (10 \text{ x } 0.05) = 3.65$$

(ب) CPI پس از بهبودها

- بهبود X (کاهش CPI دستورات Floating Point به ۵)

$$(3 \text{ x } 0.35) + (5 \text{ x } 0.20) + (4 \text{ x } 0.15) + (2 \text{ x } 0.25) + (5 \text{ x } 0.05) = 3.40$$

- بهبود Y (تقسیم دستورات Store به Fast و Slow)

سهم Fast Store: $0.15 \times 0.60 = 0.09$ با CPI=2.

سهم Slow Store: $0.15 \times 0.40 = 0.06$ با CPI=5.

$$(3 \text{ x } 0.35) + (5 \text{ x } 0.20) + (2 \text{ x } 0.09) + (5 \text{ x } 0.06) + (2 \text{ x } 0.25) + (10 \text{ x } 0.05) = 3.53$$

(ج) مقایسه و بهبود برتر

نتیجه‌گیری: بهبود X عملکرد بهتری دارد.
نتیجه:X (با 6.85٪ کاهش).

۵. **الف:**

- CPI متوسط اولیه:

$$\frac{(1 \times 20000) + (2 \times 15000) + (3 \times 5000)}{20000 + 15000 + 5000} = \frac{20000 + 30000 + 15000}{40000} = 1.625$$

- زمان کل اجرای برنامه:

$$Cycles = IC \times CPI = 40000 \times 1.625 = 65000$$

$$TotalTime = 65000 \times 0.5ns = 32.5ns$$

- پس از جایگزینی ۳۰٪ از B با D:

$$NewCPI = \frac{(1 \times 20000) + (2 \times 10500) + (3 \times 5000) + (0.5 \times 4500)}{40000}$$

$$= \frac{20000 + 21000 + 15000 + 2250}{40000} = 1.455$$

- زمان اجرای جدید:

$$40000 \times 1.455 \times 0.5 = 29.1ns$$

- بهبود عملکرد:

$$\frac{32.5}{29.1} = 1.117$$

ب:

- زمان اجرای اولیه:

$$Cycles = 100000 \times 2.0 = 200000$$

- بعد از بهینه‌سازی:

$$NewIC = 100000 \times 0.8 = 80000$$

$$NewCPI = 2.0 \times 1.1 = 2.2$$

$$NewCycles = 80000 \times 2.2 = 176000$$

- مقایسه:

$$SpeedUp = \frac{200000}{176000} = 1.136$$

- نتیجه: بهینه‌سازی سودمند است.

۶. (آ)

$$Speedup_{overall} = \frac{1}{(1 - Frac_{enhanced}) + \left(\frac{Frac_{enhanced}}{Speedup_{enhanced}}\right)}$$

$$Speedup_{overall} = \frac{1}{(1 - 0.4) + \left(\frac{0.4}{5}\right)}$$

$$= \frac{1}{0.6 + 0.08} = \frac{1}{0.68} \approx 1.47$$

(ب)

$$\frac{Frac_{enhanced}}{Speedup_{enhanced}} = \frac{0.4}{x} \to 0$$

بنابراین:

$$Speedup_{overall} = \frac{1}{(1 - 0.4) + 0}$$

$$= \frac{1}{0.6} = 1.67$$

(ج)

$$Speedup_{overall} = \frac{1}{(1 - \sum Frac_{enhanced}) + \sum \frac{Frac_{enhanced}}{Speedup_{enhanced}}}$$

$$= \frac{1}{(1 - 0.3 - 0.25) + \left(\frac{0.3}{4}\right) + \left(\frac{0.25}{3}\right)}$$

$$= \frac{1}{0.45 + 0.075 + 0.0833}$$

$$= \frac{1}{0.6083} \approx 1.64$$

۷. (آ) کد داده شده کوچکترین مقدار موجود در آرایه را پیدا می کند و با توجه این، مقدار موجود در ثبات $v1$ برابر است با 19- که همان کوچکترین عضو در آرایه داده شده است و برای ثبات $v0$ داریم 0x10010004 که آدرس همان کوچکترین عضو می باشد.

(ب) کد داده شده در اصل ۶ عضو اول آرایه را با ۶ عضو آخر جابه جا می کند پس محتوای نهایی آرایه بصورت {7, 8, 9, 10, 11, 12, 1, 2, 3, 4, 5, 6} می باشد.

(ج) کد مربوط به این بخش به صورت زیر است:

```
counter:
        li $v0, 0
loop:   andi $t0, $a0, 1
        add $v0, $v0, $t0
        srl $a0, $a0, 1
        bne $a0, $zero, loop
        jr $ra
```