درس معماری کامپیوتر

تمرین سری اول **۱۴۰۳/۱۲/۱۴**

متین باقری **۴۰۲۱۰۵۷۲۷**

---

۱) آ)

C1: 16%*6 + 10%*8 + 8%*10 + 66%*3 = 4.54 CPI

C2: 16%*20 + 10%*32 + 8%*66 + 66%*3 = 13.66 CPI

ب)

C1: $\frac{1}{4.54} \times 400 \times 10^6 \times 10^{-6} \approx 88.1\ MIPS$

C2: $\frac{1}{13.66} \times 400 \times 10^6 \times 10^{-6} \approx 29.3\ MIPS$

ج)

C1: $\frac{12000}{88.1 \times 10^6} \approx 136\ \mu s$

C2: $\frac{12000}{29.3 \times 10^6} \approx 410\ \mu s$

د)

C2': 16%*10 + 10%*8 + 8%*22 + 66%*3 = 6.14 CPI

Speed up = $\frac{13.66}{6.14} \approx 2.22$

---

۲) آ)

A: $\frac{10^9}{10^9} = 1\ CPI$

B: $\frac{1.4\times10^9}{1.2\times10^9} \approx 1.1666667\ CPI$

ب) تقریبا ٪28.57 کندتر است.

$\frac{1}{1.4} \approx 0.714$

ج)

compiler A: $10^9\ clock$

compiler B: $1.4 \times 10^9\ clock$

new compiler: $600 \times 10^6 \times 1.1 \approx 0.66 \times\ 10^9\ clock$

speed up (compared to A): $\frac{1}{0.66} \approx 1.5151$

speed up (compared to B): $\frac{1.4}{0.66} \approx 2.1212$

---

۳) آ)

MIMD (Multiple Instruction, Multiple Data) و SIMD (Single Instruction, Multiple Data) دو نوع اصلی از معماری‌های کامپیوتری هستند که در پردازش موازی استفاده می‌شوند.

SIMD: یک دستورالعمل واحد بر روی چندین داده به طور همزمان اجرا می‌شود. این معماری برای کارهایی که نیاز به پردازش موازی بر روی داده‌های مشابه دارند مناسب است. برای مثال، در پردازش تصویر یا ویدئو، یک دستورالعمل یکسان می‌تواند بر روی چندین پیکسل به طور همزمان اعمال شود. مثال: واحد پردازش گرافیکی (GPU) از معماری SIMD استفاده می‌کند. GPUها برای انجام عملیات گرافیکی که نیاز به پردازش موازی زیادی دارند، طراحی می‌شوند.

MIMD: چندین دستور مختلف بر روی چندین داده مختلف به طور همزمان اجرا می‌شود. برای کارهایی که نیاز به پردازش موازی بر روی داده‌های ناهمگن(متفاوت) دارند، مناسب است. برای مثال، در سیستم‌های چند هسته‌ای (چند پردازنده‌ای)، هر هسته می‌تواند دستورهای متفاوتی را بر روی داده‌های متفاوت اجرا کند. مثال: پردازنده‌های چند هسته‌ای از معماری MIMD استفاده می‌کنند.

تفاوت‌های اصلی:

- دستورات: در SIMD یک دستورالعمل واحد در یک لحظه دارد، در حالی که در MIMD چندین دستورالعمل مختلف در یک لحظه درحال اجرا شدن اند.
- کاربرد: SIMD برای کارهای موازی بر روی داده‌های مشابه است، ولی MIMD برای کارهای موازی بر روی داده‌های ناهمگن و نامرتبط به هم است.
- پیچیدگی: معماری MIMD پیچیده‌تر از SIMD است زیرا نیاز به مدیریت چندین دستورالعمل و داده به طور همزمان دارد.

برتری‌ها:

- SIMD: کارایی بالا در کارهای موازی بر روی داده‌های مشابه، مصرف انرژی کمتر، طراحی ساده‌تر.
- MIMD: انعطاف‌پذیری بیشتر در اجرای دستورات مختلف، مناسب برای کارهای پیچیده‌تر.

در کل این دو نوع معماری پاسخگوی دو نیاز متفاوت اند و انتخاب بین آن‌ها بسته به شرایط موجود و انتظار کاربر از کامپیوتر است.

ب) Out-of-Order Execution یک تکنیک پیشرفته در معماری کامپیوتر است که به پردازنده اجازه می‌دهد دستورات را به گونه‌ای اجرا کنند که ممکن است با ترتیبی که در کد برنامه نوشته شده‌اند متفاوت باشد. هدف اصلی این تکنیک افزایش کارایی و بهره‌وری پردازنده با استفاده بهتر از منابع موجود است.

نحوه عملکرد:

1. Fetch و Decode: پردازنده ابتدا دستورات را از حافظه Fetch کرده و آن‌ها را Decode می‌کند.

2. Dispatch: دستورات به واحد دیسپچ ارسال می‌شوند که تصمیم می‌گیرد کدام دستورات می‌توانند به طور موازی اجرا شوند.

3. Execution: دستورات به واحدهای اجرایی مختلف پردازنده ارسال می‌شوند. در این مرحله، دستورات ممکن است به ترتیبی متفاوت از ترتیب اصلی در برنامه نوشته شده اجرا شوند، به شرطی که وابستگی داده‌ها نسبت به هم رعایت شود.

4. Commit: پس از اجرا، نتایج دستورات به ترتیب اصلی برنامه ثبت (Commit) می‌شوند تا مطمئن باشیم که برنامه به درستی اجرا شده.

تأثیر بر عملکرد پردازنده:

- افزایش کارایی: با اجرای دستورات به صورت موازی و خارج از ترتیب، پردازنده می‌تواند از زمان‌های توقف (مانند انتظار برای دسترسی به حافظه یا IO devices) استفاده کند.
- کاهش تأخیر: این تکنیک به پردازنده اجازه می‌دهد تا دستورات بعدی را که وابستگی داده‌ای ندارند زودتر اجرا کند.
- بهبود استفاده از منابع: با اجرای دستورات به صورت موازی، منابع پردازنده مانند ALU و FPU بهتر استفاده می‌شوند.

چالش‌ها و محدودیت‌ها:

- پیچیدگی سخت‌افزاری: پیاده‌سازی آن نیاز به سخت‌افزار پیچیده‌تری دارد، که می‌تواند باعث افزایش مصرف انرژی و هزینه طراحی شود.
- مدیریت وابستگی‌ها: پردازنده باید به دقت وابستگی‌های داده‌ای و کنترلی بین دستورات را مدیریت کند تا از اجرای نادرست دستورات جلوگیری شود.

مثال: پردازنده‌های مدرن مانند Intel Core i7 و AMD Ryzen از این تکنیک استفاده می‌کنند تا عملکرد خود را در اجرای برنامه‌های پیچیده و multi-threaded بهبود دهند.

---

۴) آ)

$$35\% \times 3 + 20\% \times 5 + 15\% \times 4 + 25\% \times 2 + 5\% \times 10 = 3.65\ CPI$$

ب)

$$\text{X: } 35\% \times 3 + 20\% \times 5 + 15\% \times 4 + 25\% \times 2 + 5\% \times 5 = 3.4\ CPI$$

$$\text{Y: } 35\% \times 3 + 20\% \times 5 + 15\% \times (60\% \times 2 + 40\% \times 5) + 25\% \times 2 + 5\% \times 10 = 3.53\ CPI$$

ج) بهبود X عملکرد بهتری دارد و CPI را بیشتر کاهش می‌دهد و آن را از 3.65 به 3.4 می‌رساند که یک بهبود به حدود ۷ درصد می‌رسد.

$$1 - \frac{3.4}{3.65} \approx 0.06849 \approx 7\%$$

---

۵) آ)

$$\text{Average CPI: } \frac{20\times1+2\times15+3\times5}{20+15+5} = \frac{65}{40} = 1.625\ CPI$$

$$\text{CPU total time: } 1.625 \times 40000 \times 0.5 \times 10^{-9} = 32.5\ \mu s$$

$$\text{Adding class D: } \frac{20\times1+2\times15+5\times(70\%\times3+30\%\times0.5)}{20+15+5} = \frac{61.25}{40} = 1.53125\ CPI$$

$$1 - \frac{1.53125}{1.625} \approx 0.05769 \approx 6\%$$

در این صورت CPI میانگین حدود 6٪ کاهش پیدا می‌کند.

ب)

$$80\% \times 110\% = 88\%$$

بله، سودمند است و زمان اجرا را 12٪ کاهش می‌دهد.

---

۶) آ)

$$\text{Speedup} = \frac{1}{60\% + \frac{40\%}{5}} = \frac{100}{68} \approx 1.47$$

ب)

$$\lim_{x \to \infty} \frac{100}{60 + \frac{40}{x}} = \frac{100}{60} \approx 1.67$$

ج)

$$\frac{100}{45 + \frac{30}{4} + \frac{25}{3}} \approx 1.64$$

---

۷) آ)

```
array: .word 15, -19, 17, 20, -10, 12, 100, -5
       la $a0, array           # $a0 = 0x10010000
       addi $a1, $a0, 28       # $a1 = address of '-5'
       move $v0, $a0
       lw $v1, 0($v0)          # $v1 = '15'
       move $t0, $a0
loop:  addi $t0, $t0, 4        # $t0 = address of next word in the array
       lw $t1, 0($t0)          # $t1 = next word in the array
       bge $t1, $v1, skip      # if $t1 >= $v1 , skip
       move $v0, $t0           # $v0 = address of minimum number till now
       move $v1, $t1           # $v1 = minimum number till now
skip:  bne $t0, $a1, loop
```

در کد بالا کاری که در هر خط انجام می‌شود در کامنت‌ها نوشته شده. در کل در این کد روی آرایه iterate شده و کوچک‌ترین عدد و آدرس آن به ترتیب در $v1 و $v0 ذخیره می‌شود. مقدار دقیق هر ثبات در انتها به این صورت hex و decimal برابر است با:

```
$a0 = 0x10010000 = 268,500,992
$a1 = 0x1001001C = 268,501,020
$v0 = 0x10010004 = 268,500,996
$v1 = 0xFFFFFFED = -19
$t0 = 0x1001001C = 268,501,020
$t1 = 0xFFFFFFFB = -5
```

ب)

```
array: .half 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
       la $a0, array           # $a0 = address of array
       li $a1, 6               # a1 = 6, the counter
       move $t0, $a0           # $t0 = address of array, first element
       addi $t1, $a0, 12       # $t1 = address of array + 12, 6th element
loop:  lh $t3, ($t0)
       lh $t4, ($t1)
       sh $t3, ($t1)
       sh $t4, ($t0)           # swaps two elements of the array
       addi $t0, $t0, 2
       addi $t1, $t1, 2        # move pointers to next elements
       addi $a1, $a1, -1       # decrement the counter
       bne $a1, $zero , loop   # does the loop for 6 times
```

در کد بالا کاری که در هر خط انجام می‌شود در کامنت‌ها نوشته شده. ۶ عنصر اول با ۶ عنصر دوم به ترتیب جابه‌جا می‌شوند و آرایه نهایی به این صورت خواهد بود:

Array: 7, 8, 9, 10, 11, 12, 1, 2, 3, 4, 5, 6

ج)

```
.text:
       li $a0, 43              # binary =  101011  = includes 4 '1's
                    # THE FUNCTION:
       li $v0, 0
loop:
       and $t0, $a0, 1
       beq $t0, $zero, continue
       addi $v0, $v0, 1
continue:
```

```
        srl $a0, $a0, 1
        bne $a0, $zero, loop
```