



۱) از استفاده میکنیم. پس: $CPI = \sum_{i=1}^n CPI_i \times f_i$

$$CPI_{A \text{ in } P1} = 6 \times 0.4 + 6 \times 0.2 + 2 \times 0.3 + 2 \times 0.1 = 4.4$$

ب) دو برابر شدن فرکانس، مدت زمان هر چرخه کلک را نصف می‌کند، اما تعداد چرخه‌های مورد نیاز برای هر دستور (CPI) را تغییر نمی‌دهد. بنابراین، CPI مستقل از فرکانس است. اما افزایش تعداد کلک‌ها به ازای هر دستور، CPI را بیشتر میکند. بنابراین همان فرمول قبل را داریم که CPI_i چهار واحد افزایش یافته:

$$CPI_{A \text{ in } P2} = 10 \times 0.4 + 10 \times 0.2 + 6 \times 0.3 + 6 \times 0.1 = 8.4$$

ج) میدانیم زمان اجرای یک برنامه در یک پردازنده از فرمول $Time = \frac{CPI \times InstructionCount}{f_{clock}}$ محاسبه می‌شود. بنابراین:

$$Time_{A \text{ in } P1} = \frac{4.4 \times InstructionCount}{f_{clock \text{ p1}}}, Time_{A \text{ in } P2} = \frac{8.4 \times InstructionCount}{2f_{clock \text{ p1}}}$$

بنابراین پردازنده دوم بهتر است و میزان تسريع آن برابر است با:

$$SpeedUp = \frac{Time_{A \text{ in } P1}}{Time_{A \text{ in } P2}} = \frac{4.4 \times 2}{8.4} = 1.048 \Rightarrow 4.8\%$$

د) برای تصمیم گیری صرفاً CPI را در هر یک از حالت‌های گفته شده محاسبه می‌کنیم:

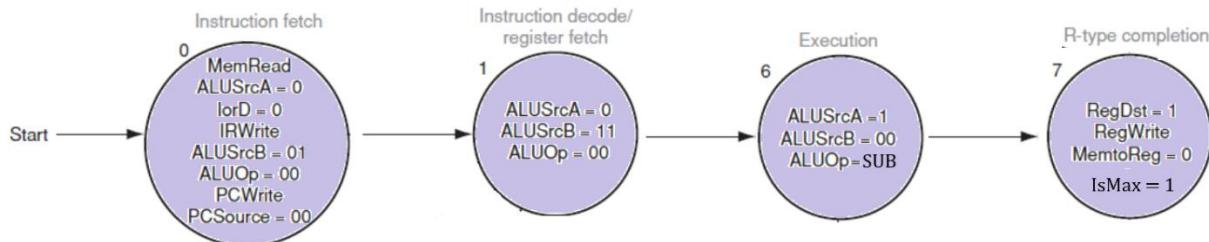
$$CPI' = 6 \times 0.4 + 6 \times 0.2 + 1 \times 0.3 + 1 \times 0.1 = 4$$

$$CPI'' = 3 \times 0.4 + 12 \times 0.2 + 2 \times 0.3 + 2 \times 0.1 = 4.4$$

همانطور که میکنید مقدار اولی کمتر از پس بهبود بهتری است.

۲) در اجرای این دستور به ALU دستور sub را میدهیم و از خروجی ALU که $rs=rt$ است استفاده می‌کنیم. اگر حالت $overflow$ را در نظر نگیریم، بیت آخر (31) نشانده‌نده علامت این تفريق است؛ اگر این بیت 1 باشد یعنی حاصل خروجی منفی بوده و مقدار rt بیشتر از rs است و در بقیه حالات این بیت 0 است. البته یک سیگنال کنترلی جدید به نام ALU از ALU خارج می‌کنیم که تنها در صورتی که دستور ما max باشد 1 می‌شود. در ادامه کافی است از دو مالتی پلکسر جدید در راه ورودی $WriteData$ استفاده کنیم. سلکت مالتی پلکسر اول سیگنال $IsMax$ است و ورودی 0 آن سیمی است که اکون به $WriteData$ وصل است و خروجی آن را به $WriteData$ وصل می‌کنیم؛ ورودی 1 آن برابر خروجی مالتی پلکسر دوم است. سلکت مالتی پلکسر دوم همان بیت آخر خروجی ALU که در رجیستر ذخیره شده است. به ورودی 1 آن مقدار rt (Read Data 1) و به ورودی 0 آن rs (Read Data 2) را وصل می‌کنیم.

ب) این دستور غالباً شبیه دستورات IF است. در IF دستورات خوانده می‌شود. در ID دستور کدگشایی می‌شود و مقادیر rs , rt خوانده می‌شوند. در مرحله EX دستور SUB روی ورودی ها اجرا می‌شود و خروجی ALU در $ALUOut$ ذخیره می‌شود. مرحله MEM را مشابه سایر دستورات $R-Type$ نداریم و در مرحله W هم مقدار $WriteData$ توسط مالتی پلکسر هایی که اضافه کردیم مشخص می‌شود و سپس در rd نوشته می‌شود. سیگنال های کنترلی را نیز در شکل کشیدم:





ج) در سیستم MIPS، دستور *la* (load address) است و در واقع یک *pseudo-instruction* (دستور ساختگی) است که برای بارگذاری آدرس یک بروچسب (*label*) در ثبات به کار می‌رود و اسمبلر آن را به مجموعه‌ای از دستورات واقعی (معمولًاً *addi* و *ori* یا *lw* و *sw*) تبدیل می‌کند. پس این دستور عملاً ۴+۴=۸ کلاک طول میکشد. برای سایر دستورات نیز تعداد کلاک‌ها مشابه اسلاید هاست. بنابراین داریم:

$$CPI = \frac{\text{Clock Cycles}}{\text{InstructionCount}} = \frac{(4 + 4) + 5 + 5 + 4 + 4 + 3 + 4 + 4}{(1 + 1) + 7} = \frac{37}{9} = 4.11$$

آ) برای ابتدا زمان مسیر بحرانی هر یک از دستورات رو پیدا میکنیم:

Instruction	Time
R-Type	$200 + 50 + 100 + 50 = 400$
<i>lw</i>	$200 + 50 + 100 + 200 + 50 = 600$
<i>sw</i>	$200 + 50 + 100 + 200 = 550$
<i>beq</i>	$200 + 50 + 100 = 350$
<i>J</i>	200

مثل همیشه *lw* بیشترین زمان را میبرد که 600 واحد زمانیست. پس در *Single-cycle* تمام دستورات در یک کلاک 600 واحدی اجرا میشوند.

در نیز ابتدا باید بیشترین زمان را بین ۵ مرحله داده شده پیدا کنیم که مشاهده میکنید این زمان متعلق به 200 واحد زمانیست. بنابراین تمام کلاک‌ها 200 واحد زمانی طول میکشند و داریم:

Instruction	Cycle count	Time
R-Type	4	$200 \times 4 = 800$
<i>lw</i>	5	$200 \times 5 = 1000$
<i>sw</i>	4	$200 \times 4 = 800$
<i>beq</i>	3	$200 \times 3 = 600$
<i>J</i>	3	$200 \times 3 = 600$

ب) میدانیم در سینگل سایکل *CPI* برابر 1 است. برای *multi* نیز از فرمول $CPI = \sum_{i=1}^n CPI_i \times f_i$ استفاده میکنیم:

$$CPI = 5 \times 0.25 + 4 \times 0.1 + 4 \times 0.52 + 3 \times 0.11 + 3 \times 0.02 = 4.12$$

ج) این تسريع را با فرمول همیشگی محاسبه میکنیم:

$$Speedup = \frac{ExecTime_{single}}{ExecTime_{multi}} = \frac{CPI_{single}}{CPI_{multi}} \times \frac{Clock cycle Time_{single}}{Clock cycle Time_{multi}} = \frac{1}{4.12} \times \frac{600}{200} = 0.728$$

هیچی دیگه مشاهده میکنیم که تسريع نداریم.

4. خوب بیاین فرض کنیم در برنامه $a\%$ از دستورات از نوع X , $b\%$ از دستورات از نوع Y و $c\%$ از دستورات از نوع Z هستند. حال هر کدام از پردازنده‌ها در این برنامه را با همان فرمول همیشگی محاسبه میکنیم.

$$CPI_{P1} = 5a + 10b + 3c$$

$$CPI_{P2} = 3a + 7b + 2c$$

$$CPI_{P3} = 2a + 5b + c$$

حال از فرمول و داده‌هایی که برای تسريع داشتیم استفاده میکنیم:

$$Speedup1 = \frac{ExecTime_1}{ExecTime_2} = \frac{CPI_1}{CPI_2} \times \frac{Clock cycle f_2}{Clock cycle f_1} = \frac{5a + 10b + 3c}{3a + 7b + 2c} \times 1.2 = 1.8 \Rightarrow a = b$$

$$Speedup2 = \frac{ExecTime_2}{ExecTime_3} = \frac{CPI_2}{CPI_3} \times \frac{Clock cycle f_3}{Clock cycle f_2} = \frac{3a + 7b + 2c}{2a + 5b + c} \times \frac{1.5}{1.2} = 1.875 \Rightarrow b = c$$

چون جمع 3 متغیر ما برابر 1 است، داریم: $a = b = c = 0.33$ و برنامه به نسبت یکسان از هر نوع دارد!



برای محاسبه کران بالا حس میکنم میشه تخمین بهتری زد ولی فعلا فقط همین به ذهنم میرسه:

$$T_{limit} \leq T_{limit} + T_{critical\ path\ 3} = \frac{1}{1.5F} \Rightarrow T_{limit} \leq \frac{2}{3F}$$

.5

آ) برای نمایش 12 تا FF به 4 بیت نیاز داریم. پس ورودی های ROM ما بیت های $OpCode$ فعالی و FF فعلی است و خروجی های آن FF بعدی و سیگنال های کنترلی است. حال سراغ محاسبه میرویم.

$$inputs: 4 + 8 = 12$$

$$output: 4 + 30 = 34$$

$$Rome\ size = 34 \times 2^{12}b = (34 \times 2^2) \times 2^{10}b = 136 \times 2^{10}b = 136\ Kb$$

ب) از ROM اول برای ایجاد سیگنال های کنترلی (30 بیت) استفاده میکنیم. برای اینکار تنها به حالت FF در ورودی نیاز داریم:

$$inputs: 4$$

$$output: 30 = 30$$

$$Rome\ size = 30 \times 2^4 = 480\ b$$

از ROM دوم برای مشخص کردن شماره FF بعدی استفاده میکنیم. برای ایجاد آن به $OpCode$ ها و FF های فعلی نیاز داریم:

$$inputs: 4 + 8 = 12$$

$$output: 4$$

$$Rome\ size = 4 \times 2^{12}b = (4 \times 2^2) \times 2^{10}b = 16 \times 2^{10}b = 16\ Kb$$

ج) طبق صورت سوال تعداد $u-instruction$ هاده اکثر 8 تاست، پس حداقل 3 بیت برای مشخص کردن آنها نیاز داریم؛ بنابراین برای

الف داریم: نقطه شروع در ROM را آدرسی میگیریم که 3 بیت کم ارزش آن صفر و 3 بیت پرازش آن $OpCode$ دستور باشد، و از این

آدرس تا 7 آدرس بعدی مربوط به این دستور خاص میشود.

$$inputs: 8 + 3 = 11$$

$$output: 30$$

$$Rome\ size = 30 \times 2^{11}b = 60\ Kb$$

حال اگر بخواهیم برای ب هم مشابه قبیل یک بار سیگنال های کنترلی و بار دیگر $state$ را مشخص میکنیم:

$$inputs: 4$$

$$output: 30 = 30$$

$$Rome\ size = 30 \times 2^4 = 480\ b$$

و رام دوم:

$$inputs: 8 + 3 = 11$$

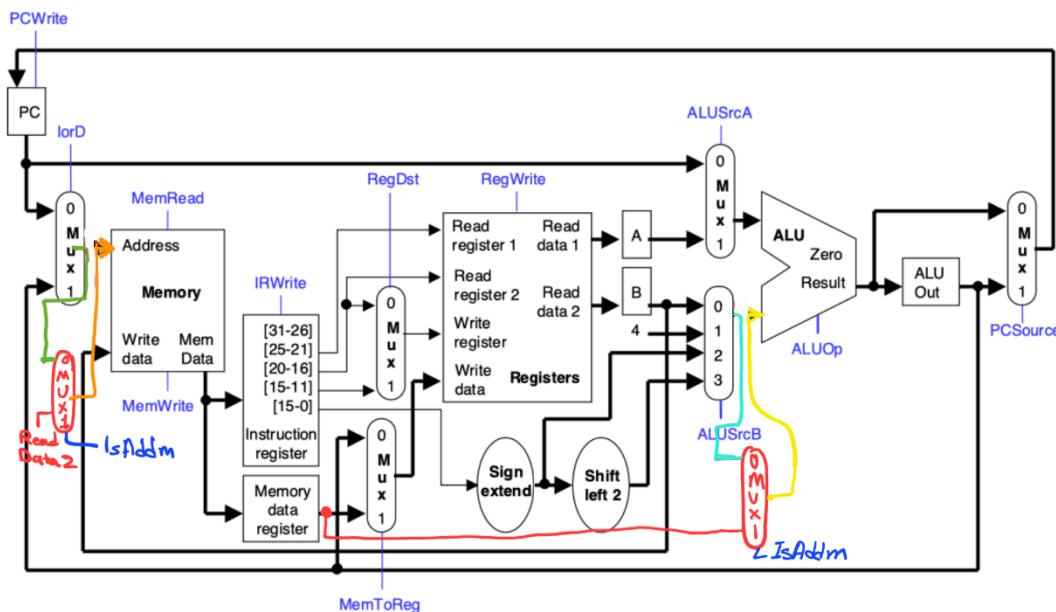
$$output: 4$$

$$Rome\ size = 4 \times 2^{11} = 8\ Kb$$

آ) مشابه تصویر به 2 تا مالتی پلکسر و یک سیگنال کنترلی جدید نیاز داریم. این سیگنال کنترلی $IsAddm$ در صورت مشاهده

این دستور 1 میشود. این 2 تا مالتی پلکسر نیز یکی برای دادن آدرس درون rt به $Memory$ و دیگری برای دادن

ALU استفاده میشود که سیمکشاوی آن را در تصویر میبینید.



ب) در تصویر حالت های جدید و سیگنال های مهم در آن حالت را مشخص کردم:

