| بسمه تعالی  پاسخنامه تمرین اول درس معماری کامپیوتر  نیمسال دوم ۰۰-۰۱ |
| --- |
| سوال اول ۳ رابطه بین throughput و latency را در نظر بگیرید:  الف) در چه صورتی میتوان گفت که این دو با یکدیگر رابطه عکس دارند؟  در صورتی که یک پردازشگر single-core داشته باشیم و پردازش موازی صورت نگیرد میتوان رابطه عکس برای این دو مورد در نظر گرفت.  ب) اگر throughput برای سیستمی افزایش یافته باشد آیا میتوان نتیجه گرفت که latency تسک ها کاهش یافته است؟ توضیح دهید.  لزوما اینطور نیست همانطور که در بخش الف گفته شد تنها در صورتی عبارت فوق صحیح است که مواردی مثل پردازش موازی نداشته باشیم و سیستم single-core باشد. اما اگر به سراغ پردازش موازی برویم و چندین core داشته باشیم ممکن است حتی با افزایش latency، افزایش throughput داشته باشیم.  ج) دو خط تولید آب آشامیدنی داریم . در خط تولید اول تعداد 2500 بطری 1.5 لیتری آب آشامیدنی در 6 ساعت بسته بندی میشود. در خط تولید دوم تعداد 10000 بطری 4 لیتری آب آشامیدنی در 48 ساعت بسته بندی میشود . شاخصی جهت مقایسه throughput این دو خط تولید ارائه کرده و throughput این دو خط را با یکدیگر مقایسه کنید.  میتوانیم throughput را برای این مسئله به شکل زیر تعریف کنیم:  تعداد بطری تولید شده در یک روز \* حجم هر بطری = throughput  خط تولید اول در 6 ساعت 2500 بطری بسته بندی میکند در نتیجه در 24 ساعت 10000 بطری بسته بندی میکند  خط تولید دوم در 48 ساعت 10000 بطری بسته بندی میکند در نتیجه در 24 ساعت 5000 بطری بسته بندی میکند  حال مقایسه را انجام میدهیم  خط تولید اول = 10000 \* 1.5 = 15000 || خط تولید دوم = 5000 \* 4 = 20000  در نتیجه خط تولید دوم با وجود اینکه تعداد بطری کمتری را بسته بندی میکند اما از جهت حجم آب معدنی در نهایت throughput بالاتری دارد. سوال دوم ۱ دنباله دستورات اجرا شده را در انواع ماشین های Accumulator ،Register-Memory و Load-Store برای اجرای دستور D= A + B - C بنویسید.  If: A+B -> E   | Accumulator | Register-Memory | Load-Store | | --- | --- | --- | | Load A  Add B  (store E, load E)  Sub C  Store D | Add A, B, E  Sub E, C, D | Load R1, A  Load R2, B  Add R3, R1, R2  Load R4, C  Sub R5, R3, R4  Store R5 |  سوال سوم ۲ الف) فرض کنید که شما یک برنامه را روی پردازنده MHz 300 خود اجرا کرده اید. با استفاده از جدول زیر CPI و MIPS را محاسبه کنید.    درحالت عادی برای محاسبه CPI صرفا کافی بود میانگین تعداد کلاک‌های لازم برای هر نوع instruction بدست اوریم با فرض اینکه فراوانی یکسانی داریم اما چون در اینجا فرکانس instruction ها با یکدیگر متفاوت است برای محاسبه CPI باید میانگین وزن داری از تعداد کلاک تمام instruction class ها بگیریم. پس داریم:  CPI = (40 \* 1) + (20 \* 1) + (10 \* 2) + (20 \* 3) + (10 \* 5)/100 = 1.9  برای محاسبه MIPS هم کافیست فرکانس کاری پردازنده را بر CPI تقسیم کنیم چرا که فرکانس در این سوال برحسب Ghz بیان شده است. پس داریم:  MIPS = 300/1.9 = 157.9  ب) فرض کنید که بهینه سازی اعمال کرده اید که 30٪ از دستورالعمل‌های arithmetic/logic (یعنی 12٪ از کل دستورالعمل ها)، 30٪ دستورالعمل های load و 20٪ از دستورالعمل های floating-point را حذف می کند. Speed up برنامه بهینه شده چقدر است؟ (محسبات را به طور کامل توضیح دهید)  فرمول پایه speedup مطابق زیر است  speedup =  روش ساده تر برای حل این سوال این است که ببینیم با این بهینه سازی CPI جدید چخ تغییری دارد پس داریم:  New CPI = 1.9 - [ (0.3 \* 0.4 \* 1) + (0.3 \* 0.2 \* 1) + (0.2 \* 0.1 \* 5) ] = 1.62  speedup = = = 1.17 |

| سوال چهارم ۲ فرض کنید بخش های a , b , c روی پردازش یک برنامه کار میکنند به گونه ای که a ، چهل درصد ابتدایی پردازش را انجام میدهد و کار را تحویل b میدهد و 25 درصد از پردازش را b انجام میدهد و تحویل c میدهد و باقی پردازش را c انجام داده و کار به اتمام میرسد . اگر سرعت b ، بیست درصد افزایش یابد و سرعت c ، سی درصد افزایش یابد ، speedUp کل را به دست آورید. حال اگر بخواهیم speedup را به 1.3 برسانیم ، سرعت کار a باید چند درصد افزایش یابد.  اگر دقت کنیم ، این مسئله مشابه مسائل دیگری است که با قانون amdahl حل میشود و صرفا صورت سوال تغییر کرده است.  1 / (0.4 + (0.25 / 1.2) + (0.35 / 1.4)) = 1.165  برای بخش دوم میخواهیم speedup را به 1.3 برسانیم در نتیجه داریم :  1 / ( (0.4 / x) +(0.25 / 1.2) + (0.35 / 1.4)) = 1.3 -> 0.4 / x = 0.31 -> x = 1.29  در نتیجه بخش a باید سرعت خود را 29 درصد افزایش دهد سوال پنجم ۳ الف) اصل مجاورت و انواع آن را توضیح دهید.  اصل مجاورت میگوید که برنامه ها در زمان اجرا در هر لحظه با بخش کوچکی از فضای آدرس خودشان کار دارند. انواع :  مجاورت زمانی : خانه هایی از حافظه که اخیراً به آنها دسترسی پیدا شده است احتمالاً به زودی دوباره مورد استفاده و دسترسی قرار می گیرند مانند اجرای حلقه ها یا استفاده از متغیرها  مجاورت مکانی : خانه هایی نزدیک به خانه های حافظه که اخیراً به آنها دسترسی پیدا کرده ایم احتمالاً به زودی مورد دسترسی قرار می گیرند مانند خانه های آرایه یا دسترسی به دستورات متوالی  ب) قطعه کدی بنویسید که انواع مجاورت در آن موجود باشد و رخ دادن مجاورت‌ها در کد را مختصرا توضیح دهید.  SUM(int\* v, int n){  int i, sum = 0  for(i=0; i<n; i++)  sum+= v[i]  }  ج) اصل مجاورت چه بهره ای برای ما دارد؟  بر مبنای اصل مجاورت ما میتوانیم سلسله مراتب حافظه را داشته باشیم تا افزایش سرعت را داشته باشیم به گونه ای که همه چیز را روی دیسک ذخیره میکنیم و مواردی را از دیسک که اخیراً به آنها دسترسی پیدا شده است (و در نزدیکی هم قرار دارند) در حافظه DRAM کوچکتری کپی میکنیم (Main Memory)  و مواردی را از Main Memory که اخیراً به آنها دسترسی پیدا کرده اید (و در نزدیکی هم قرار دارند ) ، در حافظه SRAM کوچکتر کپی میکنیم ( حافظه کش متصل به CPU )  این مسئله باعث افزایش سرعت در دسترسی و کاهش هزینه در به کارگیری حافظه های گران قیمت می شود. سوال ششم ۲ گنجایش یک حافظه اصلی 512 کیلوبایت است و اندازه هر بلوک آن دو word می باشد. در صورتی که گنجایش حافظه نهان 64 بایت بوده و سیاست جایدهی به صورت نگاشت مستقیم باشد قالب آدرس دهی خانه های حافظه را مشخص کنید. (هر word یک بایت است)  ابتدا تعداد بایت های کل حافظه را حساب می کنیم تا تعداد کل بیت های آدرس دهی مشخص شوند.  در نتیجه تعداد کل بیت های آدرس دهی 19 می باشد. با توجه به اینکه اندازه هر بلوک 16 بیت (2 بایت یا دو کلمه در این سوال) است یک بیت () برای بخش offset نیاز داریم. همین طور با توجه به اینکه هر بلوک 2 بایت است و گنجایش حافظه نهان نیز 64 بایت است، در حافظه نهان 64/2 = 32 بلوک جا می گیرد در نتیجه برای آدرس دهی حافظه نهان نیز به 5 بیت () نیاز داریم (قسمت index). در نهایت نیز 13 بیت می آدرس دهی می ماند که مربوط می شود به بخش tag.   | Tag (13bits) | Index (5bits) | Offset (1bit) | | --- | --- | --- |  سوال هفتم ۳ برنامه ای را در نظر بگیرید که درخواست های دسترسی به حافظه ی آن مانند تصویر زیر باشد (هرجا کلمه ی branch نوشته شده، یعنی دو آدرس از نظر فیزیکی در حافظه ی اصلی، پشت سر هم نیستند).    حال تصور کنید که قرار است این برنامه را روی سیستمی با ویژگی های زیر اجرا کنید :  - دارای حافظه ی نهان با 4 بلاک (هر بلاک، یک کلمه است)  - دارای یک حافظه ی اصلی با 256 بلاک  - برای نگاشت خانه های حافظه به حافظه ی نهان از نگاشت مستقیم استفاده شده است.  پس از اجرای کامل این برنامه، نرخ موفقیت (hit rate) حافظه‌ی نهان سیستم چند خواهد بود ؟  برای حل این سوال باید بلاک های فراخوانی شده را به ترتیب در نظر بگیریم و وضعیت hit یا miss آنها را بشماریم. ابتدا وضعیت فراخوانی ها برای یک دور لوپ بزرگ پنج تایی را بررسی می کنیم :  Address : 160, 161, 89, 90, 89, 90, …, 91, 219  Cache block : 0, 1, 1, 2, 1, 2, …, 3, 3  hit/miss : m, m, m, m, h, h, …, m, m  Hits : 18, Total : 24  در شروع کار که هیچکدام از داده ها داخل حافظه نهان وجود ندارند پس همه miss می شوند. وقتی به آدرس 89 و 90 می رسیم و پس از miss شدن در حافظه نهان ذخیره می کنیم، 9 بار دیگر درخواست این دو آدرس تکرار می شود در نتیجه در اینجه 9 \* 2 = 18 hit رخ می دهد. در نهایت نیز برای آدرس های 91 و 219 درخواست داده می شود که هر دو miss می شوند. پس از یک دور کامل وضعیت حافظه نهان به این صورت است :  Cache Block Address  0 160  1 89  2 90  3 219  در 4 دور بعدی لوپ بزرگ 5 تایی، آدرس 160 hit می شود. سپس چون در بلاک 1 حافظه نهان مقدار ادرس 89 وجود دارد درخواست آدرس 161 miss می شود و چون دوباره مقدار آدرس 161 در آن قرار می گیرد 89 نیز miss می شود. اما مقدار ادرس 90 را در بلاک 2 داریم پس 90 hit می شود و پس از آن نیز 9 \* 2 = 18 hit داریم. درخواست آدرس های 91 و 219 هم هردو miss می شوند.  Address : 160, 161, 89, 90, 89, 90, …, 91, 219  Cache block : 0, 1, 1, 2, 1, 2, …, 3, 3  hit/miss : h, m, m, h, h, h, …, m, m  Hits : 20, Total : 24  در نتیجه تعداد کل hit ها برابر می شود با : 20\*4 + 18 = 98  Hit Rate = 98/(24\*5) = 0.8166 |
| --- | --- | --- | --- |