گزارش کار

نوشتن یک برنامه به زبان اسمبلی MIPS

معماری کامپیوتر
استاد: دکتر دهقانی
نام دانشجو: مهدی بازیار
شماره دانشجویی:971531003

تاریخ:1399/11/11

اهداف:

- نوشتن برنامه محاسبه سن کاربر به زبان اسمبلی MIPS
- اجرای برنامه بر روی شبیه ساز

مقدمه:

درباره پردازنده های MIPS: MIPS یک معماری مجموعه دستورات RISC است که پیشتر توسط MIPS Computer Systems و حال توسط MIPS Technologies توسعه می یابد.

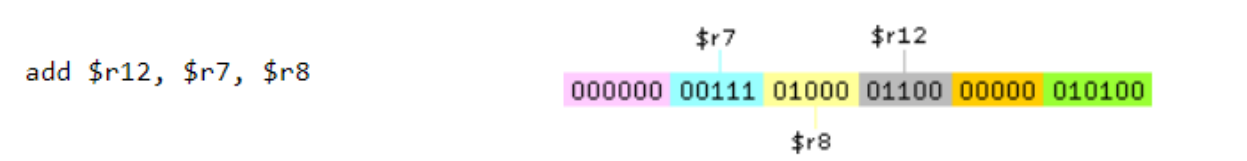
تاریخچه: پردازنده MIPS به عنوان بخشی از پروژه تحقیقاتی VLSI در دانشگاه استنفورد، در اوایل دهه هشتاد میلادی توسعه پیدا کرد. پروفسور جان هنسی، توسعه پردازنده را با یک جلسه ایده پردازی که با حضور دانشجو های فارغ التحصیل تشکیل شده بود، شروع کرد. این جلسات مطالعه و ایده پردازی، به آغاز توسعه پردازنده ای منجر شدند که از اولین پردازنده های RISC محسوب می شود.

معماری MIPS: گروه تحقیقاتی استنفورد تاریخچه ی قدرتمندی در مبحث کامپایلر ها داشتند، همین تاریخچه به آن ها کمک کرد تا پردازنده ای بسازند که معماری آن کامپایلر را خلاف عرف آن دوران، به سخت افزار نزدیک تر کند.

به همین دلیل، پردازنده MIPS، مجموعه دستورات کوچک و ساده تری را به همراه داشت. هر کدام از دستورات در یک چرخه کلاک اجرا می شدند. این پردازنده از تکنیکی به نام pipelining برای اجرای بهینه تر دستورات بهره می برد.

MIPS از 32 رجیستر که هر یک عرضی 32 بیتی دارند استفاده می کرد.

مجموعه دستورات: مجموعه دستورات MIPS از حدود 111 دستور تشکیل شده، که هر یک با 32 بیت نشان داده می شوند. مثالی از یک دستور MIPS به شکل زیر است:



بالا سمت چپ نمایش اسمبلی و سمت راست نمایش باینری دستور افزودن در MIPS اند. این دستور به پردازنده می گوید که مجموع مقادیر ذخیره شده در رجیستر های 7 و 8 را حساب کرده و نتیجه را در رجیستر 12 ذخیره کند. علامت دلار ($) برای نشان دادن یک عملیات بر روی رجیستر ها استفاده می شود. نمایش باینری رنگی سمت راست، بیانگر 6 رشته (field) یک دستور MIPS است. پردازنده نوع دستور را استفاده از اعداد باینری اولین و آخرین رشته تشخیص می دهد. در این مثال، پردازنده با دیدن صفر در اولین رشته و 20 در آخرین رشته، تشخیص می دهد که این دستور افزودن است.

عملوند ها در خانه های آبی و زرد رنگ اند و مکان دلخواه در خانه ی بنفش نمایش داده شده است. خانه ی نارنجی رنگ میزان جا به جایی (shift) را مشخص می کند، چیزی که در عمل جمع از آن استفاده نمی شود.

مجموعه دستورات شامل انواع دستورات ساده اند، از جمله:

- 21 دستور محاسباتی (+, -, *, /, %)
- 8 دستور منطقی (&, |, ~)
- 8 دستور دستکاری بیت
- 12 دستور مقایسه (>, <, =, >=, <=, ¬)
- 25 دستور انشعاب/پرس
- 15 دستور load
- 10 دستور ذخیره سازی
- 8 دستور انتقال
- 4 دستور متفرقه

MIPS در دنیای امروز: ماحصل تحقیقاتی که در دانشگاه استنفورد به ساخت نخستین جیپ MIPS منجر شد، شرکت MIPS Computer Systems در سال 1984 تاسیس شد. بعدها و در سال 1998، این شرکت توسط شرکت Silicon Graphics خریداری شده و تحت عنوان MIPS Technologies به حیات خود ادامه داد. امروزه MIPS به وسایل مختلفی در زندگی ما نیرو می بخشد.

انجام پروژه:

ابتدا دستورات SystemCall پردازنده MIPS را با هم مرور می کنیم:

System Calls

Service	Operation	Code (in $v0)	Arguments	Results
print_int	Print integer number (32 bit)	1	$a0 = integer to be printed	None
print_float	Print floating-point number (32 bit)	2	$f12 = float to be printed	None
print_double	Print floating-point number (64 bit)	3	$f12 = double to be printed	None
print_string	Print null-terminated character string	4	$a0 = address of string in memory	None
read_int	Read integer number from user	5	None	Integer returned in $v0
read_float	Read floating-point number from user	6	None	Float returned in $f0
read_double	Read double floating-point number from user	7	None	Double returned in $f0
read_string	Works the same as Standard C Library `fgets()` function.	8	$a0 = memory address of string input buffer $a1 = length of string buffer (n)	None
sbrk	Returns the address to a block of memory containing n additional bytes. (Useful for dynamic memory allocation)	9	$a0 = amount	address in $v0
exit	Stop program from running	10	None	None
print_char	Print character	11	$a0 = character to be printed	None
read_char	Read character from user	12	None	Char returned in $v0
exit2	Stops program from running and returns an integer	17	$a0 = result (integer number)	None

هر برنامه اسمبلی MIPS، دارای دو بخش .data و .text است. بخش data تمامی داده های برنامه و بخش text تمام دستورات را در خود جای می دهد.

```
1  .data
2          Month: .space 48
3          promptForYearOfBirth: .asciiz "Enter your year of birth: "
4          promptForMonthOfBirth: .asciiz "Enter your month of birth: "
5          promptForDayOfBirth: .asciiz "Enter your day of birth: "
6
7          promptForCurrentYear: .asciiz "Enter current year: "
8          promptForCurrentMonth: .asciiz "Enter current month: "
9          promptForCurrentDay: .asciiz "Enter current day: "
10
11         printAge: .asciiz "\nYou are "
12         userYear: .asciiz " year's and "
13         userMonth: .asciiz " month's and "
14         userDay: .asciiz " Day's old."
15
16
17 .text
```

خط دوم ما یک لیبل به نام Month تعریف کرده و با استفاده از عبارت .space ، مقدار 48 واحد از حافظه را به آن اختصاص داده ایم. این کار شبیه به تعریف آرایه ای در زبان اسمبلی MIPS است. دلیل 48 بایتی قرار دادن حافظه این است که ما قصد ذخیره 12 عدد از نوع int را داریم. می دانیم که هر int، 4 بایت از حافظه را اشغال می کند، پس در نتیجه برای ذخیره 12 بایت، ما به 12*4 واحد از حافظه نیازمندیم.

در خط های 2 تا 14، با استفاده از عبارت .asciiz و لیبل های مناسب، جمله های لازم برای کارکرد صحیح برنامه را ذخیره میکنیم تا در برنامه از آنها استفاده کنیم. عبارت .asciiz رشته ها را در بخش داده ذخیره کرده و یک نشانگر null هم با آن ها ذخیره می کند.

```
19         #Here we add number of days in different months
20         # Index/ Offset = $t0
21         addi $t0, $zero, 0
22         #1
23         addi $s0, $zero, 31
24         sw $s0, Month($t0)
25         addi $t0,$t0, 4
```

در این بخش از برنامه، قصد داریم که تعداد روزهای موجود در هر ماه میلادی را در آرایه ی Month که پیشتر تعریف کردیم ذخیره کنیم. برای این کار به یک offset برای پیمایش آن 48 خانه حافظه نیاز داریم. offset که در اینجا در فضای $t0 ذخیره شده، مقدار صفر دارد. پس از آن و با پر شدن هر یک از خانه های آرایه Month، مقدار offset هربار 4 واحد اضافه می شود. با دستور addi می توان مقدار 0 را با $zero جمع کرده و حاصل را که بازهم صفر است در $t0 ذخیره کرد. در خط 21، ما ابتدا با دستور addi، مقدار 31 را برای ماه اول در مکان حافظه $s0 ذخیره می کنیم (مقادیر ذخیره شده در این رجیستر در طول فراخوانی ها دست نخورده باقی می مانند.) سپس در خط 24 با استفاده از دستور sw (store word)، مقدار ذخیره شده در $s0 را در

خانه شماره $t0 آرایه Month ذخیره می کنیم. در نهایت با استفاده از دستور addi مقدار 4 واحد به عدد ذخیره شده در $t0 می افزاییم تا شماره ماه بعدی در خانه های بعدی ذخیره شود.

همه اعمال بالا تا خط 69 برنامه به منوالی که بیان شد انجام می گیرند تا در نهایت تعداد روزهای هر ماه از سال میلادی در آرایه Month ذخیره شوند.

```
73          main:
74
75          #promt the user to enter day of birth
76          li $v0, 4 #the code to display string is 4
77          la $a0, promptForDayOfBirth
78          syscall
79
80
81          #Get the user's day of birth
82          li $v0,5 #this the ins that tells the system that we want to get an int from the user
83          syscall
84          #the number is gonna be saved in v0
85
86          #store the result in $t0
87          move $t0,$v0
88
```

در خط 73، با یک لیبل مشخص کرده ایم که این بخش main برنامه ی ماست. در این بخش قصد داریم که با نشان دادن پیام هایی که در ابتدای برنامه و در بخش .data ذخیره کردیم، اطلاعات لازم برای کارکرد برنامه را از کاربر بگیریم. در ابتدا با دستور li ، دستور شماره 4 (که همانطور در جدول صفحه 3 بیان شد، برای پرینت یک رشته روی صفحه است) وارد مکان $v0 می کند. این یعنی که سیستم برای پرینت یک رشته آماده است. در ادامه با دستور la (load address) آدرس متن ذخیره شده در حافظه را وارد مکان $a0 می کنیم چرا که دستور پرینت مقداری که می خواهد پرینت شود را از این خانه بر میدارد. در نهایت در خط 78 با دستور syscall، به سیستم فرمان می دهیم که عملی که از آن خواسته ایم را انجام دهد.

در ادامه و در خط82، با استفاده از دستور شماره 5 (که طبق جدول صفحه 3 برای گرفتن یه int از کاربر است)، مقدار مورد نیاز را از کاربر دریافت کرده و سیستم آن را در مکان $v0 ذخیره می کند (مشاهده می کنید که در خط 83 هم از عبارت syscall استفاده شده تا از سیستم بخواهیم که دستورات مورد نظرمان را انجام دهد، از این به بعد این دستور در جاهای بسیاری از برنامه مورد استفاده قرار می گیرد.) برای اینکه مقدار گرفته شده از کاربر از دست نرود، در خط 87، با استفاده از دستور move آن را از مکان $v0 به مکان $t0 انتقال می دهیم.

تا خط 160 برنامه، عملیات های بالا 5 مرتبه دیگر انجام شده و مقادیر روز تولد، ماه تولد، سال تولد، روز فعلی، ماه فعلی و سال فعلی به ترتیب از کاربر گرفته شده و به ترتیب در خانه های $t0،$t1،$t2،$t3،$t4 و در نهایت $t5 ذخیره می شوند.

دستورات system call: همانطور که مشاهده کردید، تا اینجای برنامه بارها از دستور syscall، مثلا برای پرینت رشته ها، استفاده کرده ایم. پیش از اینکه هر کدام از system call های موجود در مارس را فراخوانی کنیم، کد system call در مکان $v0 ذخیره می شود. پس از آن، دستور syscall برای invoke کردن system call استفاده می شود.

در یک سیستم واقعی MIPS، دستور (instruction) syscall، یک system call exception (exception code 8) را trigger می کند که در نهایت باعث انتقال کنترل از فضای کاربر به فضای کرنل (جایی که system call هندل می شود) می شود. کرنل مقدار موجود در $v0 را بررسی کرده تا مشخص کند که کاربر کدامین system call را درخواست کرده است.

در مارس، system call ها توسط خود شبیه ساز هندل می شوند و نه توسط کرنل (exception and interrupt handler) که بتوانیم آن ها را مطالعه کرده و آن هارا تغییر دهیم. متاسفانه این باعث می شود که پیاده سازی system call ها به صورت ویرایش شده با دستور syscall غیر ممکن باشد.

```
162         j calculateAge
163
164         #end program
165         li $v0, 10
166         syscall
167
168
169         calculateAge:
```

در ادامه، برای محاسبه سن ما از یک تابع به نام calculateAge استفاده کرده ایم که دستورات محاسبه سن را در خود جای داده است. در خط 162 و در حالی که هنوز در تابع main برنامه هستیم، با استفاده از دستور j (jump unconditionally)، بدون هیچ شرطی، برنامه به خانه 169 پرش می کند. دستوری که در خط شماره 165 استفاده شده، طبق جدول برای متوقف کردن برنامه به کار می رود تا با اجرای مداوم یک دستور، برنامه با مشکل مواجه نشود. البته در این بخش برنامه چون از دستور پرش پیش از دستور توقف استفاده کرده ایم، این دستور کارایی ندارد.

```
169         calculateAge:
170         #if birth date is greater then current
171         #birth_date, then donot count this month
172         #and add 30 to the date so as to subtract
173         #the date and get the remaining days
174         bgt $t0,$t3,birthDayGreaterThanCurrentDay
175         currentDayPass:
176
177         #if birth month exceeds current month,
178         #then do not count this year and add
179         #12 to the month so that we can subtract
180         #and find out the difference
181         bgt $t1,$t4,birthMonthGreaterThanCurrentMonth
182         currentMonthPass:
```

اگر شماره روز تولد کاربر از مقدار روز فعلی بیشتر باشد، پس آن ماه محاسبه نشده و در عوض 30 روز به روز فعلی اضافه می شود تا با منهای آن ها روز های باقیمانده آن به دست آیند. در خط 174، با استفاده از شبه دستور bgt (branch if greater than)، مقادیر $t0 (روز تولد) را با $t3 (روز فعلی) مقایسه می کنیم و در صورت که این مقدار روز تولد از روز فعلی بیشتر بود، برنامه به تابع birthDayGreaterThanCurrentDay پرش می کند. لیبل ذکر شده در خط 175، برای این است که برنامه پس از اینکه تابع ذکر شده در خط بالایی را به اتمام رساند، به خط 175 از برنامه برگردد و از اینجا ادامه دهد.

خطوط 181 و 182 هم بیانگر عملیاتی شبیه به بالا هستند با این تفاوت که این بار ماه تولد و ماه فعلی با هم مقایسه شده و در صورتی که ماه تولد از ماه فعلی بیشتر بود، برنامه به تابع birthMonthGreaterThanCurrentMonth پرش کرده و پس از انجام عملیات موجود در آن تابع، به لیبل currentMonthPass بر می گردد.

```
183         #calculate date, month, year
184         sub $s0,$t3,$t0
185         sub $s1,$t4,$t1
186         sub $s2,$t5,$t2
187
188         #DisplayAgeMessage
189         li $v0,4
190         la $a0, printAge
191         syscall
192
193         #print year of age
194         li $v0,1 #to print an integer, the code is 1
195         addi $a0,$s2,0
196         syscall
197         #DisplayYearMessage
198         li $v0,4
199         la $a0, userYear
200         syscall
201
```

پس از اینکه همه عملیات بالا محاسبه شدند و مقادیر های $t0،$t1،$t2،$t3،$t4 و $t5 نهایی شدند. وقت محاسبه سن کاربر است. با استفاده از دستور sub، مقادیر روز، ماه و سال فعلی را از مقادیر روز، ماه و سال تولد کم می کنیم و به ترتیب در مکان های $s0،$s1 و $s2 ذخیره می کنیم. در نهایت با استفاده از دستوراتی که پیشتر هم در برنامه مشاهده کردیم، ابتدا متن هایی را پرینت کرده و سپس مقادیر محاسبه شده را تک تک به اطلاع کاربر می رسانیم.

```
228         birthDayGreaterThanCurrentDay:
229         sub $t1,$t1,1
230         #t6 is temporary
231         #here it saves month[birth_month - 1] value
232         sw $t6, Month($t1)
233         sub $t6,$t6,1
234         add $t3,$t3,$t6
235
236         j currentDayPass
237
238         birthMonthGreaterThanCurrentMonth:
239         sub $t5,$t5,1
240         add $t4,$t4,12
241
242         j currentMonthPass
```

در انتهای برنامه نیز دو تابع birthDayGreaterThanCurrentDay و birthMonthGreaterThanCurrentMonth را مشاهده می کنیم که عملیاتی را روی مقادیر روز و ماه و سال انجام می دهند.

برنامه فعلی در شبیه ساز Mars 4.5 نوشته شده و منبع بنده برای الگوریتم محاسبه سن، سایت GeeksForGeeks است.

در حقیقت من برنامه نوشته شده به زبان جاوا برای محاسبه سن را به برنامه ای به زبان اسمبلی میپس ترجمه کرده ام. کد اصلی جاوای این برنامه را مشاهده می کنید:

While calculating the difference in two dates we need to just keep track of two conditions that will do.

- If the current date is less than that of the birth date, then that month is not counted, and for subtracting dates we add number of month days to the current date so as to get the difference in the dates.
- If the current month is less than the birth month, then the current year is not taken into count as this year has not been completed and for getting the difference of months, we subtract by adding 12 to the current month.
- At the end we just need to subtract the days, months and years to get the difference after the two conditions are dealt with.

Below is the implementation of the above approach :

C++ | Java | Python | C# | PHP

```java
// Java program for age calculator
import java.io.*;

class GFG {
    static void findAge(int current_date, int current_month,
                        int current_year, int birth_date,
                        int birth_month, int birth_year)
    {
        int month[] = { 31, 28, 31, 30, 31, 30, 31,
                                31, 30, 31, 30, 31 };

        // if birth date is greater then current
        // birth_month, then donot count this month
        // and add 30 to the date so as to subtract
        // the date and get the remaining days
        if (birth_date > current_date) {
            current_month = current_month - 1;
            current_date = current_date + month[birth_month - 1];
        }

        // if birth month exceeds current month,
        // then do not count this year and add
        // 12 to the month so that we can subtract
        // and find out the difference
        if (birth_month > current_month) {
            current_year = current_year - 1;
            current_month = current_month + 12;
        }

        // calculate date, month, year
        int calculated_date = current_date - birth_date;
        int calculated_month = current_month - birth_month;
        int calculated_year = current_year - birth_year;

        // print the present age
        System.out.println("Present Age");
        System.out.println("Years: " + calculated_year +
              " Months: " + calculated_month + " Days: " +
              calculated_date);
    }
    public static void main(String[] args)
    {
        // present date
        int current_date = 7;
        int current_month = 12;
        int current_year = 2017;

        // birth dd// mm// yyyy
        int birth_date = 16;
        int birth_month = 12;
        int birth_year = 2009;

        // function call to print age
        findAge(current_date, current_month, current_year,
              birth_date, birth_month, birth_year);
    }
}
```