هو العلیم

اصول سیستم‌های کامپیوتری

نیمسال اول سال تحصیلی ۱۴۰۲ – ۱۴۰۱

تمرینات ۱

مریم رضائی

۱. این برنامه‌ای را که به زبان C نوشته شده است و بزرگ‌ترین مقسوم‌علیه مشترک دو عدد صحیح نامنفی را می‌یابد، به برنامه‌ای به زبان RISC-V ترجمه کنید. خودتان شماره رجیسترهایی را که در برنامه RISC-V استفاده می‌کنید طبق قراردادها تعیین کنید.

بعد از آنکه برنامه‌ی RISC-V را نوشتید، درستی آن را با ردگیری اجرای آن روی دو جفت عدد (5, 0) و (9, 21) بیازمایید.

جواب:

با فرض اینکه a در رجیستر x10 و b در رجیستر x11 قرار داشته باشند (یعنی عضو پارامترهای تابع که با آنها کار می‌کنیم) برنامه را به شکل زیر داریم (توضیح خطوط جلوی هر کدام ذکر شده است:

برنامه C	برنامه RISC-V

unsigned int gcd (unsigned int a, unsigned int b)
{
if (a = = 0 && b = = 0)
b = 1;
else if (b = = 0)
b = a;
else if (a != 0)
while (a != b)
if (a < b)
b - = a;
else
a - = b;
return b;
}
	gcd: addi sp, sp, -8 // make room in stack
	sw x10, 4(sp) // store initial value of a before it changes
	sw x11, 0(sp) // store initial value of b before it changes
	bne x10, x0, if2 // if a ≠ 0, jump to if2
	bne x11, x0, if2 // if b ≠ 0, jump to if2
	li x11, 1 // set b = 1
	j return // jump to return
	if2: bne, x11, x0, if3 // if b ≠ 0, jump to if3
	mv x11, x10 // set b = a
	j return // jump to return
	if3: beq x10, x11, return // if a ≠ 0, a = b, exit loop and return
	bltu x10, x11, inner // if a ≠ 0, a ≠ b, a < b, go to inner if
	sub x10, x10, x11 // else: a = a – b
	j if3 // retry while loop
	inner: sub x11, x11, x10 // if a ≠ 0, a ≠ b, a < b, b = b – a
	j if3 // retry while loop
	return: mv x12, x11 // return b, save it parameters reg
	lw x11, 0(sp) // get original value of b
	lw x10, 4(sp) // get original value of a
	addi sp, sp, 8 // empty stack
	ret // return using jalr x0, 0(x1)

ردگیری جفت a = 5 و b = 0 را داریم:

```
gcd:    addi sp, sp, -8  // makes room
        sw x10, 4(sp)  // stores 5
        sw x11, 0(sp)  // stores 0
        bne x10, x0, if2  // a ≠ 0 so jumps to if2
        bne x11, x0, if2
```

```
        li x11, 1
        j return
if2:    bne, x11, x0, if3 // b = 0, so continues
        mv x11, x10     // b = 5
        j return        // jumps to return
if3:    beq x10, x11, return
        bltu x10, x11, inner
        sub x10, x10, x11
        j if3
inner:  sub x11, x11, x10
        j if3
return: mv x12, x11         // sets x12 as result = 5
        lw x11, 0(sp)       // b = 0 original value
        lw x10, 4(sp)       // a = 5
        addi sp, sp, 8      // empties stack
        ret                 // returns using jalr x0, 0(x1)
```

به همین شکل ردگیری جفت a = 9 و b = 21 را داریم:

```
gcd:    addi sp, sp, -8  // makes room
        sw x10, 4(sp)  // stores 9
        sw x11, 0(sp)  // stores 21
        bne x10, x0, if2  // a ≠ 0 so jumps to if2
        bne x11, x0, if2
        li x11, 1
        j return
if2:    bne, x11, x0, if3 // b ≠ 0, so jumps to if3
        mv x11, x10
        j return
        // first run of loop:
if3:    beq x10, x11, return  // a = 9 ≠ b = 21 so continues
        bltu x10, x11, inner   // a = 9 < b = 21 so jumps to inner if
        sub x10, x10, x11
        j if3
inner:  sub x11, x11, x10     // b = 21 – 9 = 12
        j if3                 // rerun loop
        // second run of loop:
if3:    beq x10, x11, return  // a = 9 ≠ b = 12 so continues
        bltu x10, x11, inner   // a = 9 < b = 12 so jumps to inner if
        sub x10, x10, x11
        j if3
inner:  sub x11, x11, x10     // b = 12 – 9 = 3
        j if3                 // rerun loop
        // third run of loop:
if3:    beq x10, x11, return  // a = 9 ≠ b = 3 so continues
        bltu x10, x11, inner   // a = 9 > b = 3 so continues
        sub x10, x10, x11      // a = 9 – 3 = 6
        j if3                  // rerun loop
```

```
inner:   sub x11, x11, x10
         j if3
         // fourth run of loop:
if3:     beq x10, x11, return  // a = 6 ≠ b = 3 so continues
         bltu x10, x11, inner   // a = 6 > b = 3 so continues
         sub x10, x10, x11      // a = 6 – 3 = 3
         j if3                  // rerun loop
inner:   sub x11, x11, x10
         j if3
         // fifth run of loop:
if3:     beq x10, x11, return  // a = 3 = b = 3 so jumps to return
         bltu x10, x11, inner
         sub x10, x10, x11
         j if3
inner:   sub x11, x11, x10
         j if3
return: mv x12, x11            // sets x12 as result = 3
         lw x11, 0(sp)         // b = 21 original value
         lw x10, 4(sp)         // a = 9
         addi sp, sp, 8        // empties stack
         ret                   // returns using jalr x0, 0(x1)
```

بنابراین پاسخ هر دو درست بوده و برابر با پاسخ تابع اصلی و بزرگترین مقسوم‌علیه مشترک آن‌ها می‌باشد.

مراجع:

۲. این برنامه‌ای را که به زبان C نوشته شده است به برنامه‌ای به زبان RISC-V ترجمه کنید. خودتان شماره رجیسترهایی را که در برنامه RISC-V استفاده می‌کنید طبق قراردادها تعیین کنید.

با رسم شکل‌هایی، محتوای پشته را در طول اجرای تابع f1 و در طول اجرای تابع f2 مشخص کنید.

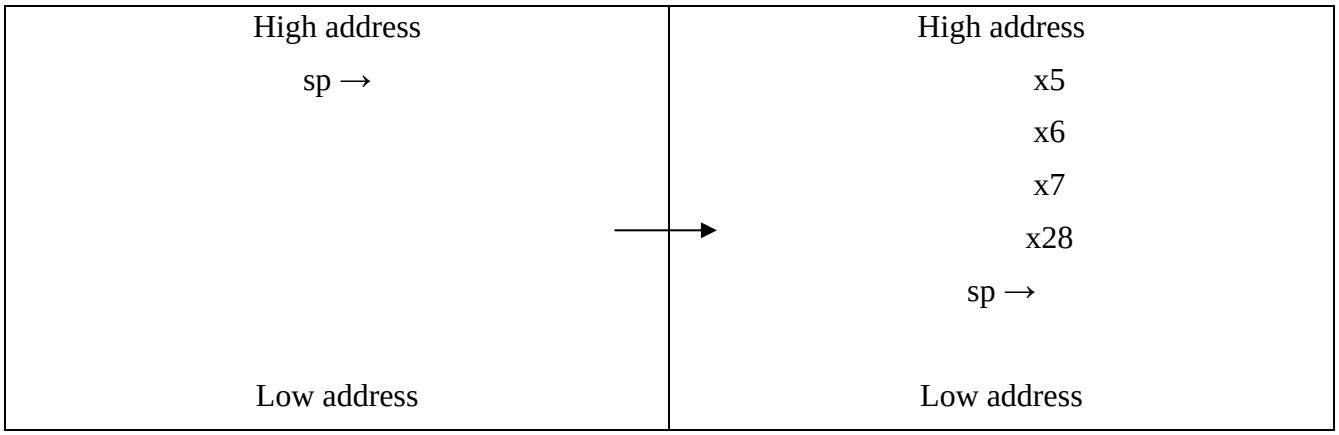
جواب:

با توجه به ذخیره مقادیر رجیسترهای مورد استفاده در تابع که میزانشان تغییر می‌کند در پشته (برای اعمال جمع و تفریق، x، i، و برای ذخیره آدرس قبلی فراخوانی تابع f1 که بتوانیم f2 را در آن فراخوانی کنیم، و همچنین r در f2) برنامه زیر را داریم. توجه می‌کنیم که a، b، p = b + i، و r + p همه پارامترهای مهم تابع‌ها هستند که در رجیسترهای x10 تا x13 آن‌ها را نگه می‌داریم.

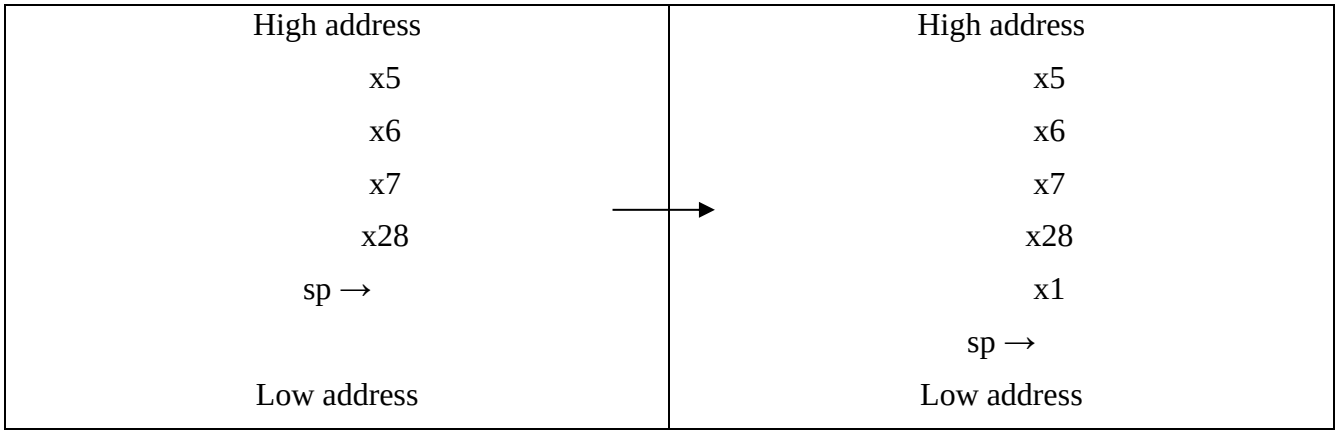
برنامه C	برنامه RISC-V

int f1(int a, int b)
{
int i, x;
x = (a + b)*(a – b);
for (i = 0; i < a; i++)
x = x + f2(b + i);
return x;
}
 	
int f2(int p)
{
int r;
r = p + 5;
return r + p;
}
	f1: addi sp, sp, -20 // make room for 5
	sw x5, 16(sp) // store previous value of x5
	sw x6, 12(sp) // store previous value of x6
	sw x7, 8(sp) // store previous value of x7
	sw x28, 4(sp) // store previous value of x28
	add x5, x10, x11 // a + b
	sub x6, x10, x11 // a – b
	mul x7, x5, x6 // x = (a + b) * (a – b)
	li x28, 0 // i = 0
	for: bge x28, x10, return // i ≥ a exit loop and go to return
	add x12, x11, x28 // b + i
	sw x1, 0(sp) // store previous call address
	jal x1, f2 // call f2 and save current address
	lw x1, 0(sp) // restore previous address
	add x7, x7, x13 // x = x + f2
	j for // retry for loop
	return: mv x14, x7 // return x, save it in parameters registers
	lw x28, 4(sp) // restore previous value of x28
	lw x7, 8(sp) // restore previous value of x7
	lw x6, 12(sp) // restore previous value of x6
	lw x5, 16(sp) // restore previous value of x5
	addi sp, sp, 20 // empty the stack
	jalr x0, 0(x1) // return to call address
	f2: addi sp, sp, -4 // make room for 1
	sw x29, 0(sp) // store previous value of x29
	addi x30, x12, 5 // r = p + 5
	add x13, x30, x12 // result = r + p, save in parameters registers
	lw x29, 0(sp) // restore previous value of x29
	addi sp, sp, 4 // empty the stack
	jalr x0, 0(x1) // return to call address

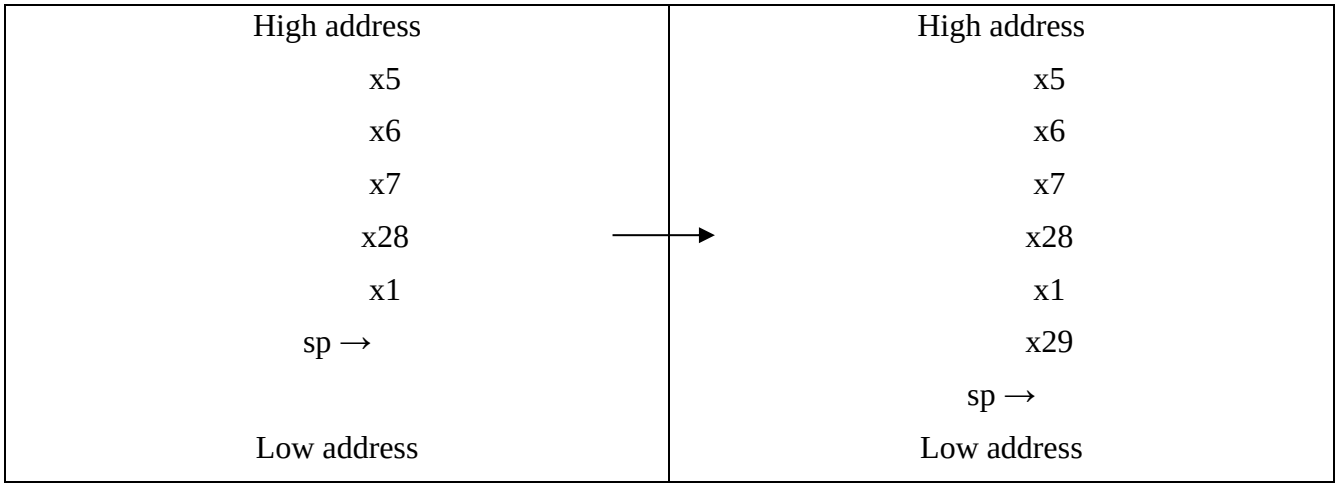
برای پشته، در ابتدای فراخوانی f1 داریم:



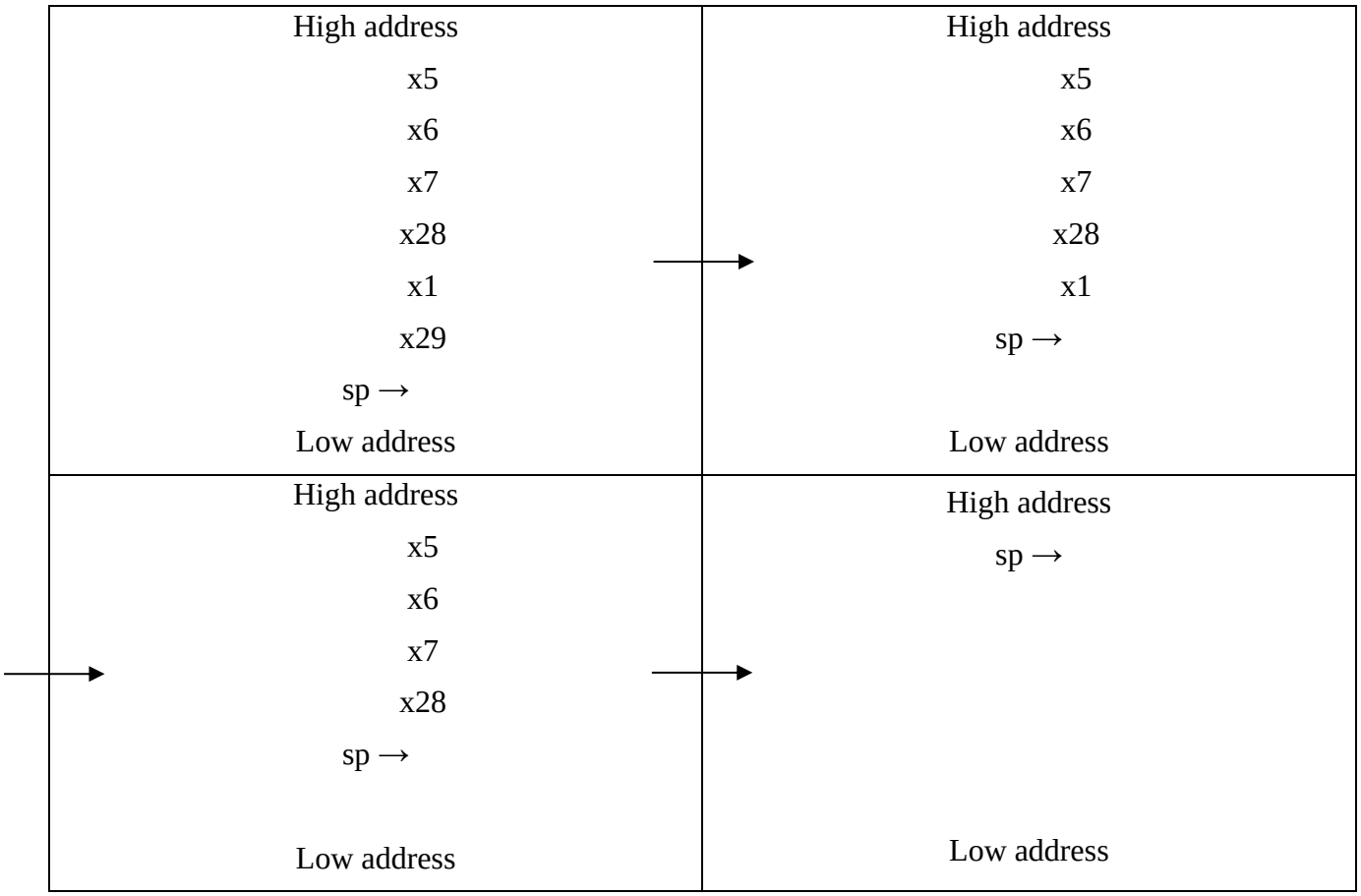
سپس برای ذخیره آدرس داریم:



حال با فراخوانی f1 در ابتدای آن داریم:



در اتمام f1، سپس بازگشت و بازیابی آدرس قبل، و سپس اتمام f2 داریم:



مراجع:

۳. این قطعه کد ماشین RISC-V را در نظر بگیرید. دستور اول در بالای بقیه دستورات قرار گرفته است.

0x01F00393
0x00755E33
0x001E7E13
0x01C580A3
0x00158593
0xFFF38393
0xFE03D6E7
0x00008067

الف) این دستورات ماشین RISC-V را به دستورات نمادین RISC-V تبدیل کنید.

ب) با مهندسی معکوس، برنامه‌ای به زبان C را که به برنامه RISC-V ترجمه شده است تعیین کنید.

پ) به فارسی روان کاری را که برنامه C می‌کند توضیح دهید. فرض کنید یکی از ورودی‌های برنامه یک عدد ۳۲ بیتی باشد که در رجیستر x10 گذاشته می‌شود و ورودی دیگر، عدد ۳۲ بیتی دیگری باشد که نشانی شروع آرایه‌ای ۳۲ عنصری از کاراکترها باشد و در رجیستر x11 گذاشته می‌شود.

جواب:

الف) با چک کردن ابتدا opcode در چپ و سپس بقیه، تبدیل زیر را داریم. توضیحات کامل در کامنت‌ها موجود است.

برنامه ماشین RISC-V	برنامه نمادین RISC-V
0x01F00393	addi x7, x0, 31 // sets initial value 31 for x7
0x00755E33	srl x28, x10, x7 // logical shifts x10 by x7 to the right // initially 31 bits, leaving only the leftmost bit
0x001E7E13	andi x28, x28, 1 // logical “and” on x28 and 0x00000001 // equals only the rightmost bit of x28
0x01C580A3	sb x28, 1(x11) // stores 8 rightmost bits of x28 in the next byte
0x00158593	addi x11, x11, 1 // adds 1 to the array start address
0xFFF38393	addi x7, x7, -1 // removes one from x7 (bits to shift x10 by)
0xFE03D6E7	bge x7, x0, -20 // goes back 5 lines if x7 is positive
0x00008067	jalr x0, 0(x1) // else returns without saving address

ب) با توجـه به خط jalr x0 در ته برنـامه RISC-V، می‌فهمیم return و یعنی یک تابـع داریم. همچنیـن به علت یکی یکی پیش رفتن مقدار x7 از ۳۱ تا ۰ با حلقه for به راحتی پیاده‌سازی می‌شود.

برنامه نمادین RISC-V	برنامه C

addi x7, x0, 31
srl x28, x10, x7
andi x28, x28, 1
sb x28, 1(x11)
addi x11, x11, 1
addi x7, x7, -1
bge x7, x0, -20
jalr x0, 0(x1)
	int store_byte(int num, int arr[])
	{
	int shift; int i = 1; int s;
	for (s = 31; s ≥ 0; s--) {
	shift = num >> s;
	shift = shift & 1;
	arr[i] = shift;
	i++;
	return arr;
	}

پ) برنامه به طور کلی عددی ۳۲ بیتی را بیت بیت از چپ در حافظه ذخیره می‌کند، به طوری که هر بیت به اندازه یک بایت جا می‌گیرد؛ به زبان دیگر عدد در حافظه به طوری ذخیره می‌شود که بین هر بیت آن ۷ صفر است. برنامه به طور دقیق تر این کار را با اعمال زیر انجام می‌دهد:

۱) در خط اول رجیستر x7 مقدار اولیه ۳۱ را می‌گیرد.
۲) عدد ۳۲ بیتی ورودی به اندازه x7 که در ابتدا ۳۱ است به راست جا به جا می‌شود و چپ آن صفر قرار می‌گیرد. یعنی برای مقدار ۳۱، تنها بیت چپی عدد باقی می‌ماند و در راست قرار می‌گیرد. برای مقادیـــر کمتر به همین شکل بیت دوم سپس سوم و غیره از چپ عدد بیت راست می‌شود.
۳) عدد حاصل با عدد یک «و» می‌شود. یعنی تنها بیت راستی حفظ شده و بیت های چپی صفر می‌شوند. یعنی مثلا اگر عدد ۳۰ بیت به راست جا به جا شده باشه و بیت دوم از چپ آن بیت راست شده بود، بیت چپی نیز که حفظ شده بود صفر می‌شود.
۴) عدد حاصل، که یک بیت از عدد ورودی‌ست (به ترتیب چپ ترین بیت، به راست) در حافظه و جایگـاه یکی بعد از شروع آرایه ذخیره شده و فضای یک بایت را می‌گیرد.
۵) به آدرس شروع آرایه کی اضافه شده و از x7 یکی کم می‌شود تا در عدد و آرایه پیش رویم.
۶) تا وقتی که x7 صفر شود به مرحله ۲ می‌رویم. در صورت کمتر از صفر شدن بدون ذخیره آدرس بازگشت داریم.

مراجع:

۴. الف) فرض کنید که دو عدد علامتدار در دو رجیستر x6 و x7 گذاشته شده باشند. دستوراتی را به زبان RISC-V بنویسید که حاصل‌جمع دو عدد را محاسبه کنند و بعد، وقوع احتمالی سرریز از رجیستر حاصل‌جمع را کشف کنند.

ب) فرض کنید دو عدد علامتدار در دو رجیستر x6 و x7 گذاشته شده باشند. دستوراتی به زبان RISC-V بنویسید که محتوای رجیستر x7 را از محتوای رجیستر x6 کم کنند و وقوع احتمالی سرریز رجیستر حاصل‌تفریق را کشف کنند.

جواب:

الف) می‌دانیم در جمع زمانی سرریز می‌تواند رخ می‌دهد که هر دو عدد منفی باشند یا هر دو مثبت باشند، زیرا در غیر این صورت (متفاوت بودن علامت‌ها) همواره پاسخ از یکی از اعداد کوچکتر خواهد بود. پس برنامه‌ای می‌نویسیم که پس از انجام جمع، علامت پاسخ را بررسی کند و برای هر علامت، هم‌علامت بودن اعداد جمع شده رو بسنجد.

```
        add x5, x6, x7            // perform addition
        bge x5, x0, check         // if result is positive, check if nums were negative
        blt x6, x0, no_overflow   // else (if result is negative), check if they were positive
                                  // if one is negative then there is no overflow
        bge x7, x0, overflow      // else (x6 was positive), if x7 is also positive then overflow
CHECK:  bge x6, x0, no_overflow   // if result was positive, and x6 also positive, then no overflow
        blt x7, x0, overflow      // else (result pos, x6 neg), if x7 is also negative then overflow
```

الف) می‌دانیم در تفریق زمانی سرریز می‌تواند رخ می‌دهد که دو عدد مختلف‌العلامت باشند. پس برنامه‌ای می‌نویسیم که پس از انجام تفریق، علامت پاسخ را بررسی کند؛ اگر پاسخ منفی بود، باید عدد اول مثبت و دوم منفی باشند تا سرریز تشخیص داده شود، زیرا علامت منهی علامت عدد دوم را نیز مثبت می‌کند و همراستا با جمع دو عدد مثبت می‌شود؛ اگر پاسخ مثبت بود، باید عدد اول منفی و دوم مثبت باشد تا سرریز تشخیص داده شود، زیرا علامت منهی علامت عدد دوم را نیز منفی می‌کند و همراستا با جمع دو عدد منفی می‌شود.

```
        sub x5, x6, x7            // perform substraction
        bge x5, x0, check         // if result is positive, check its condition
        blt x6, x0, no_overflow   // else (if result is negative), check its condition
                                  // if first one is negative then there is no overflow
        blt x7, x0, overflow      // else (x6 was positive), if second one is negative then overflow
CHECK:  bge x6, x0, no_overflow   // if result was positive, and x6 also positive, then no overflow
        blt x7, x0, overflow      // else (result pos, x6 neg), if x7 is negative then overflow
```

مراجع:

۵. الف) معماری MIPS شبیه‌ترین معماری به معماری RISC-V است. مهم‌ترین اصول مشترکی را که مبنای طراحی این دو معماری بوده است مشخص کنید.

ب) مهم‌ترین وجوه تشابه مجموعه دستورات RISC-V با مجموعه دستورات MIPS را با مثال مشخص کنید.

پ) مهم‌ترین وجوه تفاوت مجموعه دستورات RISC-V با مجموعه دستورات MIPS را با مثال مشخص کنید.

ت) توضیح دهید که معماری MIPS برای ساخت چه نوع رایانه‌هایی مناسب‌تر است و معماری RISC-V برای ساخت چه نوع رایانه‌هایی مناسب‌تر است.

جواب:

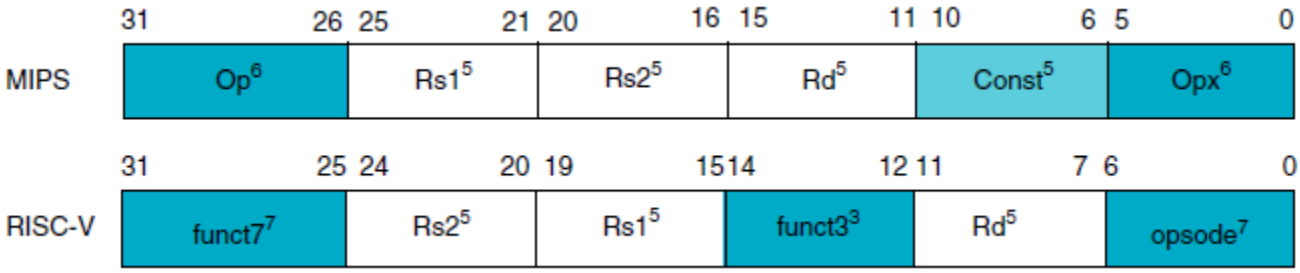
الف) معماری RISC-V و MIPS با وجود ۲۵ سال فاصله در شکل گیریشان، فلسفه یکسانی دارند و از دستورات نمادین حول دسته‌ها و ساختار دودویی یکسانی استفاده می‌کنند. مهمترین اصول این شباهت در معماری به شرح زیرند:

- تمام دستورات هر دو معماری طول ۳۲ بیت دارند.
- هر دو ۳۲ رجیستر با کاربردهای عمومی دارند که یکی از آن‌ها مختص عدد صفر است.
- در این دو زبان تنها راه دسترسی به حافظه توسط دستورات load و store است.
- هر دو بر خلاف بعضی معماری‌ها، هیچ دستوری که توان بارگیری و ذخیره چند رجیستر را داشته باشد ندارند.
- هر دو دستورات شرطی‌ای برای برابر بودن و برابر نبودن با صفر دارند.
- در این معماری‌ها، هردو دسته حالات آدرس‌دهی برای تمامی اندازه‌های کلمات کار می‌کنند.

ب) دستورات MIPS شباهت بسیاری به دستورات RISC-V دارند. در دستورات نمادین آن‌ها نام و فرمت استفاده در اکثر دستورات یکی هستند. برای مثال:

RISC-V	MIPS
add x5, x6, x7	add $1, $2, $3
addi x5, x6, 100	addi $1, $2, 100
mul x5, x6, x7	mul $1, $2, $3
and x5, x6, x7	and $1, $2, $3
andi x5, x6, 100	andi $1, $2, 100
sll x5, x6, x7	sll $1, $2, 10
lw x5, 100(x6)	lw $1, 100($2)
sw x5, 100(x6)	sw $1, 100($2)

و غیره، که در تمامی آن‌ها اسم نمادین دستور و تعداد رجیسترهای استفاده شده یکسان بوده و ترتیب آن‌ها نیز یک معنا دارند. یعنی برای مثال، در دستور جمع RISC-V حاصل جمع x6 و x7 در x5 ذخیره شده و در MIPS حاصل جمع $2 و $3 در $1 ذخیره می‌شود. همچنین در فرمت دودویی دستورات (که برای هر دو طول ۳۲ می‌باشد) در بسیاری از موارد شباهت چینش زیادی مشاهده می‌کنیم. برای مثال:



که در آن رجیسترهایی که عمل بر آن‌ها انجام شده و همچنین موارد تعیین کننده دستور اکثرا زیر هم بوده و همه هم‌اندازه‌اند از راست چیده شده‌اند.

پ) از تفاوت‌های اصلی MIPS و RISC-V در دستورات مقایسه آن‌هاست. دستورات مقایسه اصلی دو معماری را داریم:

RISC-V	MIPS
slt $1, $2, $3	slt $1, $2, $3
slti $1, $2, 100	slti $1, $2, 100
beq x5, x6, 100	beq $1, $2, 100
bne x5, x6, 100	bne $1, $2, 100
blt x5, x6, 100	
bge x5, x6, 100	

همانطور که می‌بینیم، RISC-V برای مقایسه بزرگی و کوچکی و سپس شاخه‌بندی دستوراتی ارائه می‌دهد. در حالی که MIPS تنها دستورات شاخه‌ای برای برابری و عدم برابری داشته و برای مقایسه دو رجیستر نیاز است شرطی را از پیش بررسی کرده و رجیستری را برابر ۰ یا ۱ قرار دهیم؛ آنگاه MIPS می‌تواند با دو دستور بالا و برسی ۰ یا ۱ بودن رجیستر، عملیات شاخه کردن را انجام دهد. برای مثال، شبه دستور blt $2, $3, 100 را می‌توان با دستورات زیر پیاده‌سازی کرد:

```
slt $1, $2, $3          // if $2 < $3 then $1 =1 else $1 = 0
bne $1, $0, 100         // if $1 ≠ 0 then go to PC+100
```

لازم به ذکر است که MIPS از دستورات مقایسه جز برابری تنها slt را داشته و فقط می‌تواند مقایسه کوچکتر بودن را انجام دهد. پس برای مقایسه بزرگتر بودن یا باید ترتیب یا شرط را تغییر داد و یا شبه دستور bge و bgt را با دستورات بالا پیاده‌سازی کرد. به طور کلی دستورات MIPS فلسفه ساده‌گرایانه داشته و مجموعه آن‌ها از RISC-V بزرگتر است.

ت) معماری RISC-V یک معماری رایج، ساده، رایگان و متن‌باز است که می‌تواند به راحتی پشتیبانی شود، گسترش یابد، و برای برنامه‌های خاص شخصی‌سازی شود. این معماری به سازنده‌های دستگاه‌های کوچک‌تر این اجازه را می‌دهد که به راحتی و به طور رایگان سخت افزار بسازند، و در سیستم‌های زمینه‌ای با کارایی بالا، سرورهای ابری، کنترل‌ها، پردازنده‌های عمومی، و زمینه‌های هوش مصنوعی و یادگیری ماشین و یارانش مرزی کاربرد دارد. البته به علت متن‌باز بودن، طراحی پردازنده‌ای را شامل نبوده و خود باید پردازنده‌ای توسعه دهد یا بخرد و هزینه‌ای جدا برای پیاده‌سازی و توسعه دارد.

اما معماری MIPS در حین ساده بودن، فضای محدودتر و نیاز به خرید پروانه داشته و آزادی بالای معماری RISC-V برای شخصی سازی یا پیشرفت را ندارد. همچنین به علت محدودیت اعمال شرطی و نیاز به شبه دستورها، پیاده‌سازی‌های با کارایی بالا ناکارمد می‌شوند. به این علت در دستگاه‌های کوچک مانند روترها کاربرد دارد. البته به علت وجود پروانه قابل خرید، برنامه‌ها، سخت‌افزارها و پشتیبانی تضمین شده بوده و هزینه‌ی جدای مالی یا زمانی برای پیاده‌سازی ندارد.

مراجع:

کتاب منبع درس:

Computer Organization and Design (RISC-V Edition) – Patterson & Hennessy

اینترنت:

https://www.forbes.com/sites/tiriasresearch/2018/04/05/what-you-need-to-know-about-processor-architectures/?sh=35a017c04f57
https://www.elprocus.com/risc-v-processor/
https://www.fierceelectronics.com/embedded/what-risc-v