تمرین تئوری پنجم

سید محمدرضا جوادی 402105868

ساختار کامپیوتر، دکتر اسدی

پاییز 1403

سوال1)

این بار خط به خط پیش می روم برعکس تمارین قبل، زیرا فرایند کال کردن ها پیچیده تر شده و نمی توان به سادگی تشخیص دهید من در تحلیل کجای کار هستم و گم می کنید.

```
.text
main:
    lui $a0, 1024
```

در 16 بیت بالایی و در 11 امین بیت بینشان 1 می گذارد. یعنی مقدار ورودی برای سابروتین می شود 0x04000000

```
    jal func1
```

تابع 1 را فراخوانی می کند، ادامه توضیح در خط اول

```
    addi $a0, $a0, 1
    li $v0, 1
```

می خواهیم a0 را چاپ کنیم

```
    Syscall
```

A0 چاپ می شود.

```
    li $v0, 10
```

کد ترمینیت کردن برنامه و پایان دادن آن

```
    syscall
```

تامام

```
func1:
```

این تابع کال شد

```
    xori $a0, $a0, 1
```

ورودی را با 1 xorکرد. A0 می شود 0x04000001

```
    j func2
```

به لیبل تابع دو بدون هیچ شرط می پرد ادامه آنجا

```
    addi $a0, $a0, 1
    jr $ra

func2:
    xori $a0, $a0, 1
```

دوباره با یک ایکسور می شود. مقدار ورودی می شود 0x04000000

```
    addi $ra, $ra, 4
```

به مقدار آدرس بازگشت 4 تا اضافه می کند یعنی به نوعی ریترن آدرس به یک خط بعد ارجاع می شود و یکی می پرد.

```
    lw $s0, func2 + 36($zero)
```

مقدارخط 9 ام بعد از فانک2 یعنی jal func3 در s0 ریخته می شود. در واقع 9 خط بعد از باز کردن شبه دستورات یعنی:
00000011111000000000000000001000

```
    xor $s0, $s0, $a0
```

دستور تغییر داده می شود با تغییر بیت 5ام از 0 به 1 که دستوری j است و دیگر ra تغییر نمی کند.

```
    sw $s0, func2 + 36($zero)
```

دستور جدید را قرار می دهد

```
    jal func3
```

به تابع 3 می رویم.

```
    jr $ra

func3:
    srl $a0, $a0, 26
```

ورود را 26 بیت به راستی شیفت می دهیم. A0 می شود:

0x00000001

```
add $a0, $a0, $a0
```

ورودی را با خودش جمع می کنیم، a0می شود:
0x00000002

```
jr $ra
```

به دو خط بعد از فراخوانی تابع دو در تابع یک باز می گردیم.

سوال2)

الف)

هرکدام بر اساس 6 بیت اول، اگر رجیستر فرمت بود 6 بیت آخر فانکت و گرنه نگاه می کنیم کدام فرمت است و از دستور العمل ها حل می کنیم.

```
0x20080000
addi $t0, $zero, 0
0x00044820
add $t1, $zero, $a0
0x200A0001
addi $t2, $zero, 1
0x010A4020
add $t0, $t0, $t2
0x214A0001
addi $t2, $t2, 1
0x112A0001
beq $t2, $t1, 1
0x08100003
j 0x100003
```

اما کار ترجمه ادامه دارد:

آدرسی که در آن قرار گرفته به شکل زیر ترجمه می شود

0000 0100 0000 0000 0000 0000 1100

از آنجا که آدرس خط اول

00000100 0000 0000 0000 0000 0000

است پس یعنی یک لیبل در 3 خط بعد یعنی خط چهارم داشتیم

همچنین در خط قبلی آن 1 در immediateقرار دارد که ضرب در 4 شود با 4 هم جمع شود یعنی به نوعی دو خط جلو رود یا کلا حلقه تمام می شود. در نهایت پس کد ترجمه شده به شکل زیر درمی آید:

```
addi $t0, $zero, 0
add $t1, $zero, $a0
addi $t2, $zero, 1
label: add $t0, $t0, $t2
addi $t2, $t2, 1
beq $t2, $t1, secondLabel
j label
secondLabel:
```

ب)

هربار t0 را یک واحد افزایش می دهد هرگاه برابر با مقدار ورودی بود حلقه متوقف می شود. اگر مقدار a0 مثبت بود که تا a0 می آید و درنهایت برنامه پایان می یابد. اگر منفی باشد هیچگاه متوقف نمی شویم

ج)

همان که در ب مطرح شد. برنامه در صورت منفی بودن ورودی هیچگاه پایان نمی یابد.

د)

می توان شرطی گذاشت که در صورت منفی بودن a0 حلقه اجرا نشود یا بهر نحو منفی بودن ورودی را هندل کند

ه)

دستوراتی هستند که PC را تغییر می دهند. می توانند آدرس نسبی، مطلق دهند یا با تبدیل مقدار داده شده در PC آدرس را ذخیره کنیم.

می توانیم با شبه دستور پرش blt مشکل منفی بودن را حل کنیم.

```
addi $t0, $zero, 0
add $t1, $zero, $a0
addi $t2, $zero, 1
blt $t1, $zero,secondLabel
label: add $t0, $t0, $t2
addi $t2, $t2, 1
beq $t2, $t1, secondLabel
j label
secondLabel:
```

سوال3)

دستور ها:

setdivis $rd, $rs, $rt

```
    div $rs, $rt
    mfhi $at
    beq $at, $zero, div
    add $rd, $zero, $zero
    j end
div:
    andi $rd,$rd,0
    ori $rd, $zero, 1
end:
```

bitcount $rd, $rs

```
ori $rd, $rd, 0
count_loop:
    andi $at, $rs, 1
    add $rd, $rd, $at
    srl $rs, $rs, 1
    bne $rs, $zero count_loop
```

circularlshift $rd, $rs, shift

```
ori $at, 32
lui $rd, 0x0000
ori $rd, shift
div shift, shift
mflo $t2
andi $at, $rs, 0xFFFFFFFF
sll $rd, $at, shift
srl $at, $at, (32 - shift)
or $rd, $rd, $at
```

max $rd, $rs, $rt

```
slt $at, $rs, $rt
beq $at, $zero, set_rd_to_rs
add $rd, $rt, $zero
j end_max
set_rd_to_rs:
add $rd, $rs, $zero
end_max:
```

سوال4)

```
.globl main
.data
array: .word 5, 2, 8, 1, 6  # Example array
array_size: .word 5

.text
main:
    la $t0, array
    lw $t1, array_size
    li $t2, 0

outer_loop:
    bge $t2, $t1, end_outer_loop
    li $t3, 0

inner_loop:
    sub $t4, $t1, 1
    bge $t3, $t4, end_inner_loop

    sll $t5, $t3, 2
    add $t5, $t5, $t0
    lw $t6, 0($t5)
    lw $t7, 4($t5)

    ble $t6, $t7, no_swap
    sw $t7, 0($t5)
    sw $t6, 4($t5)
```

```
no_swap:
    addi $t3, $t3, 1
    j inner_loop

end_inner_loop:
    addi $t2, $t2, 1
    j outer_loop

end_outer_loop:
    li $v0, 10                  # Exit program
    syscall
```

سوال5)

الف)

نیاز داریم تا یک بیت دیگر به opcode اضافه کنیم. طول دستورات یک بیت افزایش می یابد و از این به بعد opcode 7تایی می شود

ب)

در همان حالت اول باز به افزایش یک بیتی نیاز داریم. علاوه بر آن:

حال که 64 رجیستر در کل داریم 6 بیت برای آدرس دهی نیاز داریم. در دستورات r format طول دستورات 3 بیت افزایش می یابد. در I format 2 تا و دیگری نیاز ندارد. برای اینکه تعداد بیت ها یکسان بماند همه را به تعداد 4 تا زیاد می کنیم یعنی همه دستورات 36 بیتی می شوند تا هم نیاز های خواسته شده سوال تامین شود هم risc بودن معماری رعایت شود.

ج)

برای 76 opcode به 7 بیت نیازمندیم. برای آدرس دهی رجیستر ها به 5 بیت برای هریک نیازمندیم. حداقل اندازه هر رجیستر باتوجه به اندازه حافظه باید4 مگا باشد، نیازی نیست به هربیت اشاره کنیم، پس 22 بیت باید باشد تا 2 به توان 22 را پوشش دهد.

حداقل طول می شود 7+6+6+6 یعنی 25 بیت

سوال6)

خیلی دیر تغییرات در گروه اعلام شد و من فرصت تغییر نداشتم.با همان توضیحات cw که سند است و pdfپی که گذاشتن در cw این جواب من است.

کد اسمبلی:

```
fun:
    bne a0, $0, norm
    sllv t0, a1, a2
    bne t0, $0, norm
    li v0, 0
    jr ra
norm:
    sw ra, 0(sp)
    subi sp, 4
    li t0, 32
```

```
    sub t0, t0, a2
    sllv t0, a0, t0
    srlv a0, a0, a2
    srlv a1, a1, a2
    or a1, a1, t0
    jal fun
    addi v0, v0, 1
    addi sp, 4
    lw ra, 0(sp)
    jr ra
```

کد زبان c

```c
int fun(int a, int b, int c){
    int t = b << c;

    if (a == 0 && t == 0)
        return 0;

    t = 32 - c;
    t = a << t;
    a = a >> c;
    b = b >> c;
    b = b | t;

    return fun(a, b, c) + 1;
}
```

خروجی به ازای fun(0, 5, 1) برابر 3 است و برای fun(1, 2, 3) برابر یک.