معماری کامپیوتر دکتر عطارزاده

تمرین سری چهارم – سوال اول

با استفاده از جدول زیر و جدول داده شده درانتهای کتاب، دستور ها را اسمبل می‌کنیم.

31:25	24:20	19:15	14:12	11:7	6:0	
funct7	rs2	rs1	funct3	rd	op	R-Type
$imm_{11:0}$		rs1	funct3	rd	op	I-Type
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op	S-Type
$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op	B-Type
$imm_{31:12}$				rd	op	U-Type
$imm_{20,10:1,11,19:12}$				rd	op	J-Type
fs3	funct2	fs2	fs1	funct3	fd	op	R4-Type
5 bits	2 bits	5 bits	5 bits	3 bits	5 bits	7 bits	

Figure B.1 RISC-V 32-bit instruction formats

Table 6.1 RISC-V register set

Name	Register Number	Use
zero	x0	Constant value 0
ra	x1	Return address
sp	x2	Stack pointer
gp	x3	Global pointer
tp	x4	Thread pointer
t0–2	x5–7	Temporary registers
s0/fp	x8	Saved register/Frame pointer
s1	x9	Saved register
a0–1	x10–11	Function arguments/Return values
a2–7	x12–17	Function arguments
s2–11	x18–27	Saved registers
t3–6	x28–31	Temporary registers

```
0xA0028 Func1:    addi t4, a1, 0
```
-1

الف)

Imm = 0000 0000 0000

Rs1 = 01011

Funct3 = 000

Rd = x29 =11101

Op =0010011

0000 0000 0000 0101 1000 1110 1001 0011 = 0x00058E93

ب) این دستور از نوع I-Type است و از 2 رجیستر و یک immediate استفاده می‌کند.

Type = I-Type

Addressing mode: Immediate Addressing

```
ori  a0, a0, 32
```
-2

Imm = 0000 0010 0000

Rs1 = x10 = 01010

Funct3 = 110

Rd = 01010

Op = 0010011

00 0 0000 0101 0110 0101 0001 0011 = 0x2056513

Type = I-Type

Addressing mode = Immediate adddressing

```
sub  a1, a1, a0
```
-3

Funct7= 0100 000

Rs2 = 01010

Rs1 = 01011

Funct3=000

Rd = 01011

Op=0110011

0100 0000 1010 0101 1000 0101 1011 0011 = 0x40A585B3

Type = R-Type

Addressing mode = Register – Only addressing

```
jal   Func2
```

- 4

0xA0058 - 0xA0034 =0x00024 = 0000 0000 0000 0010 0100

Imm = 0 00000100100000000000

Rd = 00001

Op = 1101111

0000 0010 0100 0000 0000 0000 1110 1111 = 0x024000EF

Type = J-Type

Addressing mode = PC-relative mode

```
lw    t2, 4(a0)
```

- 5

Imm =0000 0000 0000 0100

Rs1= 01010

Funct3 =010

Rd =00111

Op=0000011

0000 0000 0000 0100 0101 0010 0011 1000 0011 = 0x00452383

Type = I-Type

Addressing mode = Base addressing

```
sw    t2, 16(a1)
```
-6

Imm = 0000 0001 0000

Rs1=01011

Rs2=00111

Funct3=010

Op= 0100011

0000 0000 0111 0101 1010 1000 0010 0011 = 0x0075A823

Type = S-Type

Addressing mode = Base addressing

```
srli t3, t2, 8
```
- 7

Imm = 0000 0000 1000

Rs1=00111

Rd=11100

Funct3=101

Op=0010011

0000 0000 1000 0011 1101 1110 0001 0011 = 0x0083DE13

Type = I-Type

Addressing mode = Immediate addressing

```
beq  t2, t3, Else
```
-8

Imm =0xA006C-0xA0064 = 0x00008 = 0 0000 0000 1000

Rs1 =00111

Rs2=11100

Funct=000

Op=110 0011

00000001110000111000010001100011 = 0x01C38463

Type = B-Type

Addressing mode = PC-Relative Addressing

```
jr   ra
```
-9

Jr ra == jalr x0, x1, 0

Imm = 0000 0000 0000

Rd = 00000

Rs1= 00001

Funct3=000

Op=1100111

00000000000000001000000001100111 = 0x00008067

Type =I -Type

Addressing mode = immediate Addressing

```
addi a0, a0, 4
```
-10

Imm = 0000 0000 0100

Rs1=01010

Rs2=01010

Opt=0010011

Funct3=000

00000000010001010000010100010011= 0x00450513

Type = I-Type

Addressing mode = Immediate Addressing

```
j     Func2
```
-11

J Func2 = jal x0, -24

Imm = 111111101000

Rd = 00000

Op = 1101111

11111110100111111111000001101111 = 0x FE99F06F

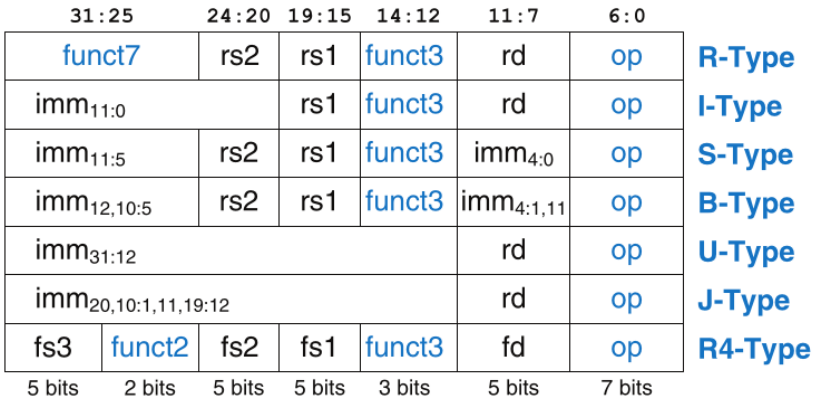
Type = J-Type

Addressing mode = PC-Relative Addressing

معماری کامپیوتر - دکتر عطارزاده

تمرین سری ۴ - سوال ۲

برای تبدیل هر کدام از دستورات ماشین به دستورات Assembly، ابتدا آن را با نوشتار باینری می‌نویسیم. سپس به ۷ بیت اول یا همان opcode نگاه می‌کنیم. از روی opcode، می‌توان نوع دستور را شناسایی کرد. بعد از شناسایی نوع دستور، در صورت نیاز به مقادیر func3 و func7 نیز نگاه می‌کنیم. از ترکیب opcode و func3 و func7 (در صورت وجود)، دستور اسمبلی مورد نظر پیدا می‌شود. همچنین با توجه به مشخص بودن نوع دستور، مقادیر rd و rs1 و rs2 و همچنین مقدار immediate موجود در دستور را نیز می‌توان پیدا کرد.



به طور مثال، برای دستور اول به شکل زیر عمل می‌کنیم:

```
0xff810113
```

آن را به صورت باینری می‌نویسیم:

111111111000 00010 000 00010 0010011

در بالا، opcode که با رنگ آبی مشخص شده، برابر 0010011 است، که نشان می‌دهد دستور از نوع I-Type است. سپس با بررسی مقدار func3 که با رنگ قرمز مشخص شده و برابر 000 است، مشخص می‌شود که کد بالا مربوط به دستور addi است:

```
addi ...
```

حال می‌توانیم مقادیر rd و rs1 و imm را پیدا کنیم:

111111111000 00010 000 00010 0010011

در نتیجه:

```
rd  = x2 (sp)
rs1 = x2 (sp)
imm = -8
```

و در نهایت، دستور به صورت زیر به دست می‌آید:

```
addi sp, sp, -8
```

با بررسی کردن دستورات دیگر به صورت بالا، برنامه کامل به دست می‌آید (برای پیدا کردن دستورات از روی opcode، از جدول ضمیمه شده در کتاب H&H استفاده شده است):

```
1
0xff810113
111111111000 00010 000 00010 0010011
addi sp, sp, -8

2
0x00a12223
0000000 01010 00010 010 00100 0100011
```

```
sw a0, 4(sp)

3
0x00112023
0000000 00001 00010 010 00000 0100011
sw ra, 0(sp)

4
0x00100293
000000000001 00000 000 00101 0010011
addi t0, zero, 1

5
0x00a2c863
0000000 01010 00101 100 10000 1100011
imm = 0x10 = 16
blt t0, a0, 0x10

6
0x00100513
000000000001 00000 000 01010 0010011
addi a0, zero, 1

7
0x00810113
000000001000 00010 000 00010 0010011
addi sp, sp, 8

8
0x00008067
000000000000 00001 000 00000 1100111
jalr zero, ra, 0
ret

9
0xfff50513
111111111111 01010 000 01010 0010011
addi a0, a0, -1

10
0xfddff0ef
```

```
11111101110111111111 00001 1101111
imm = 11111111111111011100 = -36
jal ra, -36

11
0x00412303
000000000100 00010 010 00110 0000011
lw t1, 4(sp)

12
0x00012083
000000000000 00010 010 00001 0000011
lw ra, 0(sp)

13
0x00810113
000000001000 00010 000 00010 0010011
addi sp, sp, 8

14
0x02a30533
0000001 01010 00110 000 01010 0110011 (R-type)
mul a0, t1, a0

15
0x00008067
000000000000 00001 000 00000 1100111
jalr zero, ra, 0
ret
```

کد بالا، تابعی بازگشتی است که فاکتوریل تنها آرگمان خود را حساب می‌کند. این کد به شکل زیر عمل می‌کند:

```
factorial:
addi sp, sp, -8
sw a0, 4(sp)
sw ra, 0(sp)
```

این قسمت از کد، به اندازه ۲ رجیستر (۸ بایت) جا در stack رزو می‌کند و رجیسترهای a0 و ra را در stack ذخیره می‌کند تا مقدار آن از دست نرود. (در بالا، لیبل factorial برای توضیح بهتر در ادامه کد اضافه شده است).

```
addi t0, zero, 1
```

```
blt t0, a0, 0x10
```
این قسمت، شرط 1 < a0 را بررسی می‌کند. در صورتی که شرط برقرار باشد، به اندازه ۱۶ بایت (یعنی ۴ دستور) به پایین می‌پرد.

```
addi a0, zero, 1
addi sp, sp, 8
ret
```
در صورتی که شرط قبلی برقرار نباشد، این قسمت اجرا می‌شود. این قسمت از کد، مقدار 1 را به عنوان مقدار بازگشتی تابع برمی‌گرداند و همچنین فضای اشغال شده stack را آزاد می‌کند.

```
addi a0, a0, -1
jal ra, -36
```
این قسمت از کد، factorial(n-1) را صدا می‌زند (با فرض این که n همان آرگومان تابع فعلی است). برای صدا زدن factorial، کافیست به اندازه ۳۶ بایت (یا ۹ دستور) به عقب بپریم تا به ابتدای تابع برسیم.

```
lw t1, 4(sp)
lw ra, 0(sp)
addi sp, sp, 8
mul a0, t1, a0
ret
```
در نهایت، چون مقدار n (ورودی تابع) را در مرحله قبل عوض کردیم، مقدار اولیه آن را از stack لود می‌کنیم و در t1 ذخیره می‌کنیم. همچنین مقدار اولیه ra را نیز از stack لود می‌کنیم و در نهایت، مقدار

```
n * factorial(n - 1)
```
محاسبه می‌شود (مقدار factorial(n-1) را قبلا حساب کرده بودیم و در رجیستر a0 ذخیره شده است). در خط آخر نیز این مقدار را برمی‌گردانیم.

می‌توان کد C زیر را معادل کد بالا درنظر گرفت:

```
int factorial(int n) {
     if (n <= 1) return 1;
     else return n * factorial(n - 1);
}
```

معماری کامپیوتر دکتر عطارزاده

تمرین سری چهارم – سوال سوم

ابتدا با دیرکتیو .data array base address را مشخص می‌کنیم. سپس تابع های داده شده در صورت سوال را به زبان اسمبلی برمی‌گردانیم.

```
.data
arr: .space 160 # array has 160(40*4) bytes
```

مقادیر اولیه را در رجیستر های دلخواه می‌ریزیم.

```
4 .text
5 li s0, 40 # s0 keeps max_size
6 la s1, arr # s1 keeps arr
7 li s2, 14 # s2 keeps inp
8 li s3, -1 # s3 keeps ans
```

و مطابق صورت سوال بقیه کد را اسمبل می‌کنیم.

```
j main

func:
    bgt a0, zero, else1 # if (a0 == 0)
    li a0, 0
    ret # return 0

    else1:
    li t0, 1
    bne a0, t0, else2 # if (a0 == 1)
    li a0, 1
    ret # return 1

    else2:
    slli t0, a0, 2
    add t0, s1, t0 # t0 = s1 + 4*a0 (the address of arr[i])

    lw t1, 0(t0)
    beq t1, zero, else3 # if (arr[i] != 0)
    add a0, t1, zero
    ret # return arr[i]

    else3:
    addi sp, sp, -16 # reserve stack space for 4 registers
    sw t0, 0(sp)
    sw ra, 4(sp)
    sw a0, 8(sp)

    addi a0, a0, -1
    jal func # call func(a0 - 1)

    lw ra, 4(sp)
    slli t1, a0, 1 # t1 = 2 * func(a0 - 1)

    lw a0, 8(sp)
    sw t1, 12(sp)

    addi a0, a0, -2
    jal func # call func(a0 - 2)
```

```
50
51      lw t0, 0(sp)
52      lw ra, 4(sp)
53      lw t1, 12(sp)
54
55      sub t2, a0, t1 # t2 = a0 - t1 = (func(a0 - 2)) - (2 * func(a0 - 1))
56      sw t2, 0(t0) # arr[i] = t2 (= func(a0 - 2) - 2 * func(a0 - 1))
57      add a0, t2, zero
58      addi sp, sp, 16 # release stack space
59      ret # return arr[i]
60
61  main:
62      li t0, 0
63      for1: # setting array 0
64          bge t0, s0, endfor1
65          slli t1, t0, 2
66          add t1, t1, s1
67          sw zero, 0(t1)
68          
69          addi t0, t0, 1
70          j for1
71      endfor1:
72
73      addi s5, s0, -1
74      for2:
75          blt s5, zero, endfor2
76          add a0, s5, zero
77          jal func
78
79          bgt a0, s2, else
80          add s3, s5, zero
81          j endfor2
82
83          else:
84
85          addi s5, s5, -1
86          j for2
87      endfor2:
88  end:
```

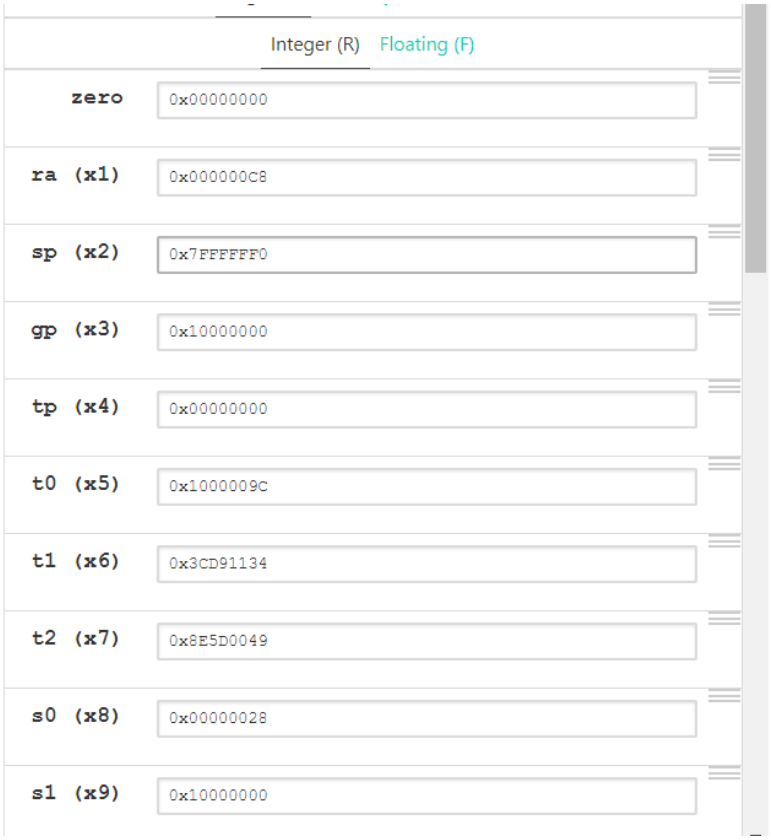
اعضای آرایه:

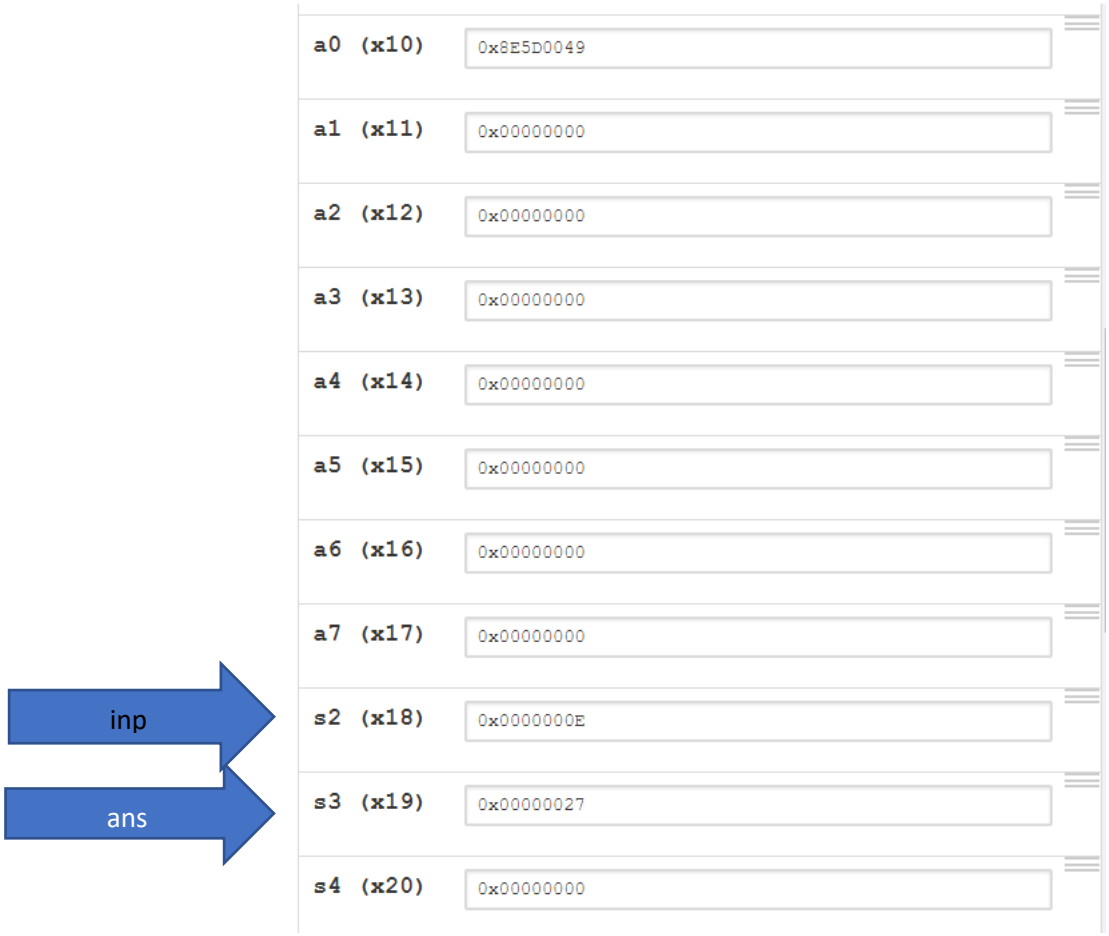
(این اعضا توسط نرم افزار Rars مشخص شده اند.)

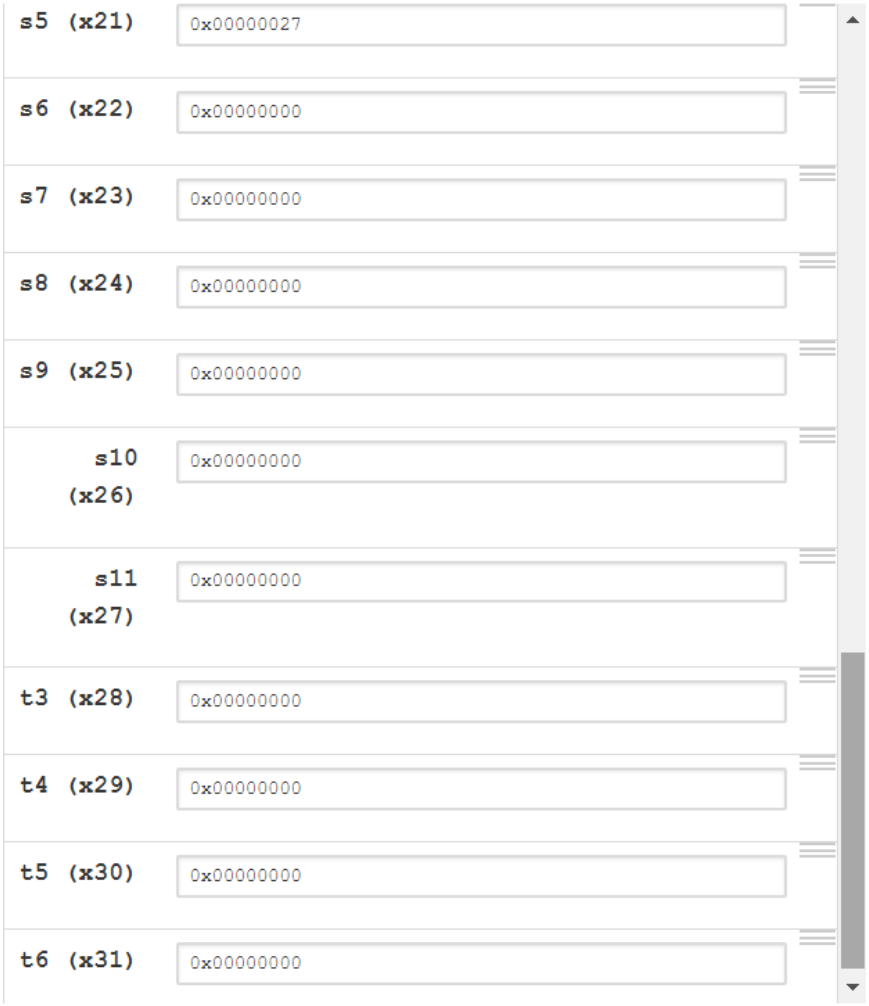
Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0	0	-2	5	-12	29	-70	169
0x10010020	-408	985	-2378	5741	-13860	33461	-80782	195025
0x10010040	-470832	1136689	-2744210	6625109	-15994428	38613965	-93222358	225058681
0x10010060	-543339720	1311738121	1128151334	-944564547	-1277686868	1610809189	-204337950	2019485089
0x10010080	51659168	1916166753	514292958	887580837	-1260868716	-885649027	510429338	-1906507703
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0
0x10010100	0	0	0	0	0	0	0	0
0x10010120	0	0	0	0	0	0	0	0

رجیستر ها پس از احرای برنامه:

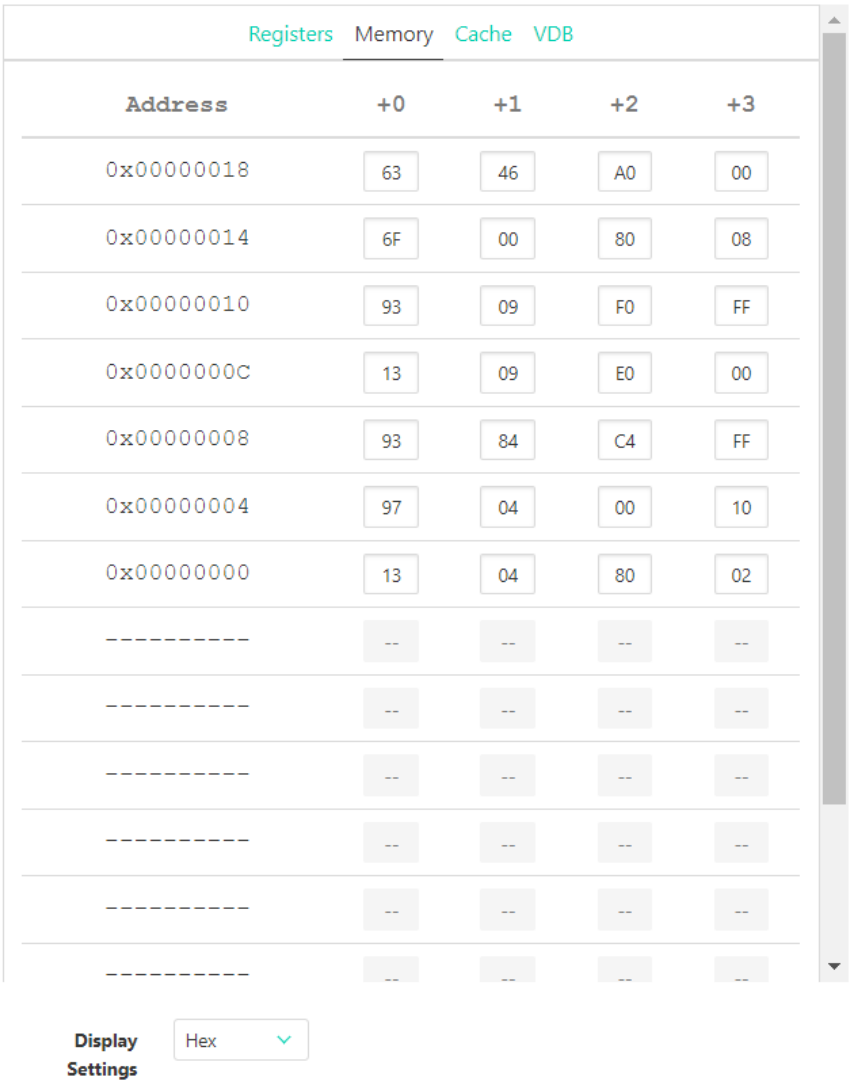



a0 (x10)
0x8E5D0049
a1 (x11)
0x00000000
a2 (x12)
0x00000000
a3 (x13)
0x00000000
a4 (x14)
0x00000000
a5 (x15)
0x00000000
a6 (x16)
0x00000000
a7 (x17)
0x00000000
s2 (x18)
0x0000000E
s3 (x19)
0x00000027
s4 (x20)
0x00000000
inp
ans



لازم به ذکر است که با inp=14، جواب 39 شد که در رجیستر s3 قرار دارد.

ممووری:


Registers
Memory
Cache
VDB
Address
+0
+1
+2
+3
0x00000018
63
46
A0
00
0x00000014
6F
00
80
08
0x00000010
93
09
F0
FF
0x0000000C
13
09
E0
00
0x00000008
93
84
C4
FF
0x00000004
97
04
00
10
0x00000000
13
04
80
02
Display Settings
Hex