Session2

Topics

1. Simple recurrsive function (S.R-func) → convert c program to RISC-V assembley
2. Reccursive sum (R-sum) → copmute sum of 1, 2, …, n that n is input
3. -Sum of matrix elements
4. Palindrom → put an array on mem then check if its palindrom
5. R-fib → find n-th element of fib series when n is input
6. Convert

S.R-func

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;
}	

// X10=a, X11=b, X12=x, X5=i

S.R-func

// X10=a, X11=b, X12=x, X5=i

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;
}
	Int f1:
	add x6, x10, x11 //x6=a+b, x6= temp register
	sub x7, x10, x11 //x7=a-b, x7= temp register
	
	add x12, x6, x7 // x=x6+x7
	addi x5, x0, 0 //i=0

// X10=a, X11=b, X12=x, X5=i

برنامه C	برنامه RISC-V
int f1(int a, int b) { int i, x; x = (a + b)+(a - b); →	Int f1: add x6, x10, x11 //x6=a+b, x6= temp register sub x7, x10, x11 //x7=a-b, x7= temp register add x12, x6, x7 // x=x6+x7 addi x5, x0, 0 //i=0

for (i = 0; i < a; i++) →
x = x + f2(b + i);
return x;
 	
}
	addi x2, x2, -4 // adjust stack for an item
	sw x1, 0(x2) // store x1(return addres) on stack
	Forloop:
	bge x5, x10, Exit // if a>= i jump to Exit
	add x14, x11, x5 // p=b+i
	jal x1, intf2 // jump to f2
	add x12, x12, x15 //x=x+f2(b+i)
	addi x5, x5, 1 //i++
	beq x0, x0, Forloop // repeat for loop
	
	Exit:
	lw x1, 0(x2) // restore the return address
	addi x2, x2, 4 // adjust stack pointer to pop 1 item
	jalr x0, 0(x1) //jump to main program

int f2 (int p) →
{
int r;
r = p + 5;
return r + p;
}
	intf2:
	addi x13, x14, 5 //r=p+5
	add x15, x13, x14 //x15=r+p
	jalr x0, 0(x1) // jump to f1

کد اجرایی S.R-func

```
addi x10, x0, 10
addi x11, x0, 8
addi sp, sp, 400
beq x0, x0, Start
Int f1:
   add x6, x10, x11
   sub x7, x10, x11
   add x12, x6, x7
   addi x5, x0, 0

     addi x2, x2, -4
     sw x1, 0(x2)
 Forloop:
     bge x5, x10, Exit
     add x14, x11, x5
     jal x1, intf2
     add x12, x12, x15
     addi x5, x5, 1
     beq x0, x0, Forloop

intf2:
  addi x13, x14, 5
  add x15, x13, x14
  jalr x0, x1, 0

 Exit:
```

```
    lw x1, 0(x2)
    addi x2, x2, 4
    jalr x0, x1, 0
Start:
   jal x1, Int f1
```

R-sum

برنامه C	برنامه RISC-V

int sum(int a){
 	
if (a>1) →
return a+sum(a-1);
else
return 1;
}
	addi x10, x0, 7 // x10-> nth
	addi sp , sp, 400
	beq x0, x0, End //start computing sum function
	
	F:
	
	L:
	
	End:
	jal x1, F

R-sum

برنامه C	برنامه RISC-V

int sum(int a){
 	
if (a>1) →
return a+sum(a-1);
else
return 1;
}
	addi x10, x0, 7 // x10-> nth
	addi sp , sp, 400
	beq x0, x0, End //start computing sum function
	
	F:
	addi sp, sp, -8 //adjust stack for two elements
	sw x1, 4(sp) // store return address
	sw x10, 0(sp) //store n on stack
	
	addi x5, x10, -1 //x5=n--
	bge x5, x0, L // if (n-1)>=0 go to L, else n==0 so load previos n from stack
	addi sp , sp, 8
	jalr x0, x1, 0
	
	L: // n -=1 then store on stack
	addi x10, x10, -1 //n-=1
	jal x1, F // store n-- on stack
	
	addi x6, x10, 0 // store x10 on temp regisster x6
	lw x10, 0(sp) // load previos x10 and return add from stack
	lw x1, 4(sp)
	addi sp, sp, 8
	
	add x10, x10, x6 //x10= sum of previos and present value x10
	jalr x0, x1, 0
	End:
	jal x1, F

Sum of matrix elements

*arr در x10 و در هر سطر ماتریس 10 تا درایه داریم پس برای انتقال هر سطر به سطر بعدی با توجه نحوه ذخیره سازی ماتریس باید .... تا به شمارش گر اضافه کنیم

برنامه C	برنامه RISC-V

float mat_mean (int arr[][])
{
  int i, j;
  float mean, sum = 0;
 	
  for (i = 0; i < 10; i++)
  for (j = 0; j < 10; j++)
  sum += arr[i][j];
  mean = sum / 100;
  return mean;
}
	Float mat:
	  addi x5, x0, 0 // i=0
	  addi x6, x0, 0 //j=0
	  addi x7, x0, 0 // sum=0
	  addi x11, x0, 0 // mean=0
	  addi x29, x0, 10 //x29=10

Sum of matrix elements

برنامه C	برنامه RISC-V

float mat_mean (int arr[][])
{
  int i, j;
  float mean, sum = 0;
 	
  for (i = 0; i < 10; i++) →
  for (j = 0; j < 10; j++) →
 	
  sum += arr[i][j]; →
 	
  mean = sum / 100; →
  return mean;
}
	Float mat:
	  addi x5, x0, 0 // i=0
	  addi x6, x0, 0 //j=0
	  addi x7, x0, 0 // sum=0
	  addi x11, x0, 0 // mean=0
	  addi x29, x0, 10 //x29=10
	
	  loop1:
	  bge x5, x29, exit //if i>=10 go to exit
	  loop2:
	  bge x6, x29, reset //if j>=10 then j=0
	
	  mul x28, x5, x29 // x28= i*10
	  add x28, x28, x6 //x28= x28+j
	  slli x28, x28, 2 // x28*4
	  add x28, x28, x10 // x28= x28+ *arr
	  lw x30, 0(x28) //lw arr[i][j]
	  add x7, x7, x30 //sum+= arr[i][j]
	
	  addi x6, x6, 1 // j++
	  beq x0, x0, loop2 //repeat loop2
	
	reset:
	  addi x6, x0, 0 // j=0
	  addi x5, x5, 1 //i++
	  beq x0, x0, loop1 // repeat loop1
	
	exit:
	  addi x29, x29, 90 //x29=100
	  div x11, x7, x29 //mean= sum/100

کد اجرایی محاسبه مجموع تمام عناصر ماتریس 2*2

```
// i=x5
//j=x6
// sum=x7 →final out put
//x29= matrix dim=2
// x3= input for multipication

// x4 = input for multipication

//x31 = output for multipication


                    //→جای گذاری ماتریس در حافظه

addi sp, sp,400

add x10, sp, x0


addi x5, x0, 10

sw x5, 0(x10)

addi x10, x10, 4


addi x5, x0, 2

sw x5, 0(x10)

addi x10, x10, 4


addi x5, x0, 5

sw x5, 0(x10)

addi x10, x10, 4


addi x5, x0, 8

sw x5, 0(x10)


addi x10, x10,-12      */
```

```
beq x0, x0, start
Float mat:                  //مقدار دهی اولیه متغییر ها
  addi x5, x0, 0
  addi x6, x0, 0
  addi x7, x0, 0
  addi x29, x0, 2           */

  loop1:
     bge x5, x29, exit                  //if i>=10 go to exit
     loop2:
        bge x6, x29, reset              //if j>=10 then j=0

          mulx3, x4:                    // x28= i*10
           add x3, x0, x5
           add x4, x0, x29
           jal x1, multiple
           add x28, x0, x31
           addi x31, x0, 0              */

        add x28, x28, x6                //x28= x28+j
        slli x28, x28, 2                // x28*4
        add x28, x28, x10               // x28= x28+ *arr
        lw x30, 0(x28)                  //lw arr[i][j]
        add x7, x7, x30                 //sum+= arr[i][j]

        addi x6, x6, 1                  // j++
        beq x0, x0, loop2               //repeat loop2
```

```
reset:
   addi x6, x0, 0                    // j=0
   addi x5, x5, 1                    //i++
   beq x0, x0, loop1                 // repeat loop1

multiple:                            //store registers
   addi sp , sp, -20
   sw x7, 16(sp)
   sw x8, 12(sp)
   sw x9, 8(sp)
   sw x10, 4(sp)
   sw x11, 0(sp)                     */

addi x10, x0, 16                     // compute multiplication
addi x7, x0, 1
add x8, x0, x0
beq x0, x0, Wexp

Wloop:
    and x9, x4, x7
    beq x9, x7, Multi
 Endif:
    addi x8, x8, 1
    slli x7, x7, 1
    beq x0, x0, Wexp
   Multi:
      sll x11, x3, x8
      add x31, x31, x11
      beq x0, x0, Endif
```

```
  Wexp:
    blt x8, x10, Wloop                    */
    lw x11, 0(sp)                        //recover registers
    lw x10, 4(sp)
    lw x9, 8(sp)
    lw x8, 12(sp)
    lw x7, 16(sp)
    addi sp, sp, 20                   */

    jalr x0, x1, 0

start:
   jal x1, Float mat
exit:
```

Palindrom

آدرس شروع آرایه در حافظه در رجیستر x10 قرار دارد(پارامتر تابع).

طول رشته نیز در رجیستر x11 قرار دارد.

اگر متقارن باشد مقدار x9=1 در غیر اینصورت 0 است.

برنامه RISC-V

راه1)

```
addi x9, x0, 0
addi x11, x11, x10                //x11= string length+ array address
palindrom:
   beq x11, x10, Equal            // palindrom checked
   lw x8, 0(x10)                  // load first element of array (load a byte)
   lw x6. 0(x11)                  // load last element of array (load a byte)
   bne x6, x8, Nequal             // if x8!= x6 then branch
   addi x10, x10, 1               //x10++ (go to next byte)
   addi x11, x11, -1              //x11—(go to next byte)
   beq x0, x0, palindrom

Equal:
      addi x9, x0, 1
Nequal:
```

کد اجرایی آرایه متقارن:

```
addi x10, x0, 400          // جای گذاری آرایه5 کلمه ای در حافظه
addi x5, x0, 97
sw x5, 0(x10)

addi x5, x0, 98
addi x10, x10, 4
sw x5, 0(x10)

addi x5, x0, 99
addi x10, x10, 4
sw x5, 0(x10)

addi x5, x0, 98
addi x10, x10, 4
sw x5, 0(x10)

addi x5, x0, 97
addi x10, x10, 4
sw x5, 0(x10)              */

addi x10, x0, 400           //x10=starst array address
addi x11, x0, 16            //x11= end array address
```

```
addi x9, x0, 0
add x11, x11, x10
palindrom:
   beq x11, x10, Equal
   lw x8, 0(x10)
   lw x6, 0(x11)
   bne x6, x8, Nequal
   addi x10, x10, 4
   addi x11, x11, -4
   beq x0, x0, palindrom

Equal:
      addi x9, x0, 1

Nequal:
```

R-fib

// x10=n= input for Int fib

برنامه C	برنامه RISC-V
int fib(int n) {	addi x11, x0, 0 //x11=0 * addi x12, x0, 0 //x12=0 addi x5, x0, 1 //x5=1 addi x10, x0,7 // x10=n addi sp, sp, 400 // sp+=400 beq x0, x0, End //start function *
if (n == 0)	Int fib:
return 0;	
else if (n == 1)	L:
return 1;	
Else	
return fib(n−1) + fib(n−2); →	End: //jump to Int fib
}	Jal x1, Int fib

برنامه C	برنامه RISC-V

int fib(int n) {
 	
if (n == 0) →
return 0;
else if (n == 1)
return 1;
else
return fib(n−1) + fib(n−2);
}
	addi x11, x0, 0 //x11=0 = output *
	addi x12, x0, 0 //x12=0
	addi x5, x0, 1 //x5=1
	addi x10, x0,7 // x10=n
	addi sp, sp, 400 // sp+=400
	beq x0, x0, End //start function *
	
	Int fib: //inpute =x10
	addi sp, sp, -4 // adjust stack for an item **
	sw x1, 0(sp) // save return address **
	
	blt x5, x10, L // if n>=2 jump to L (*)
	beq x10, x5, 24 // if n==1 return 1 else n==0 (***)
	addi x12, x0, 0 //x12 =0 =return val
	lw x1, 0(sp) // load last return address
	addi sp, sp, 4 //adjust sp to pop 2 item
	add x11, x11, x12 //add x12 to total value
	jalr x0, x1, 0 // jump to caller
	
	//n==1//
	addi x12, x0, 1 //x12=1 because n==1 (***
	lw x1, 0(sp) //load return address
	addi sp, sp, 4 //adjust sp to pop 2 item
	add x11, x11, x12 // add x12 to total value
	jalr x0, x1, 0 // jump to caller ***)
	
	L: //copmute fib for (n-1) & (n-2)
	addi x10, x10, -1 // n=n-1
	addi sp, sp, -4 // adjust stack for an item (*
	sw x10, 0(sp) // save x10 to stack *)
	jal x1, Int fib //run Int fib(n-1)
	lw x10, 0(sp) // load n-1 /*
	addi sp, sp, 4 //adjust sp to pop 2 item */
	addi x10, x10, -1 //n=n-2
	jal x1, Int fib //run fib(n-2)
	//now both fib(n-1)&(n-2) finished//
	
	lw x1, 0(sp) // load return address
	addi sp, sp, 4 //adjust sp to pop 2 item
	jalr x0, x1, 0 // jump back to caller
	
	End: //jump to Int fib
	Jal x1, Int fib

برنامه RISC-V	توضیح
addi x11, x0, 0 //x11=0	X11 که مقدار نهایی خروجی تابع است را صفر میکنیم ←
addi x12, x0, 0 //x12=0	X12 همیشه Fib(0) یا Fib(1) را نگهداری میکند صفر میکنیم ←
addi x5, x0, 1 //x5=1	X5 عدد 1 را برای مقایسه های موجود در خود نگهداری میکند ←
addi x10, x0,7 // x10=n	مقدار n را در رجیستر x10 قرار میدهیم ←
addi sp, sp, 400 // sp+=400	مقدار sp را افزایش میدهیم تا در زمان اجرا مقدار منفی اختیار نکند ←
beq x0, x0, End //start function	به برچسب End در انتها میرویم تا در زمان فراخوانی آخر برنامه پایان پذیرد. ←
Int fib:	
addi sp, sp, -4 // adjust stack for an item	پشته را برا ذخیره آدرس محل فراخوانی آماده میکنیم ←
sw x1, 0(sp) // save return address	آدرس محل فراخوانی را در پشته ذخیره میکنیم ←
blt x5, x10, L // if n>=2 jump to L	اگر n 0یا 1 نبود به برچسب L میرویم تا آن بازگشتی محاسبه شود ←
beq x10, x5, 24 // if n==1 return 1	تشخیص میدهیم n برابر 1 است یا نه در صورت تساوی 1 را برمیگردانیم←
addi x12, x0, 0 //x12 =0 =return val	در این خط تشخیص داده ایم n مساوی 0 بوده پس x12 را صفر میکنیم←
lw x1, 0(sp) // load last return address	حال آدرس محل فراخوانی را بازیابی کرده ←
addi sp, sp, 4 //adjust sp to pop 2 item	پشته را آزاد کرده ←
add x11, x11, x12 //add x12 to total value	مقدار Fib محاسبه شده را با خروجی کلی(x11) جمع میکنیم ←
jalr x0, x1, 0 // jump to caller	به محل فراخوانی تابع برمیگردیم ←
addi x12, x0, 1 //x12=1 because n==1	تشخیص داده ایم n=1 پس باید مقدار x12 را برابر 1 کنیم ←
lw x1, 0(sp) //load return address	حال آدرس محل فراخوانی را بازیابی کرده ←
addi sp, sp, 4 //adjust sp to pop 2 item	پشته را آزاد کرده ←
add x11, x11, x12 // add x12 to total value	مقدار Fib محاسبه شده را با خروجی کلی(x11) جمع میکنیم ←
jalr x0, x1, 0 // jump to caller	به محل فراخوانی تابع برمیگردیم ←
L:	پس n بزرگ تر از 1 بوده و باید مقدار آن به صورت بازگشتی محاسبه شود.
addi x10, x10, -1 // n=n-1	n را یک واحد کم میکنیم تا Fib(n-1) محاسبه شود ←
addi sp, sp, -4 // adjust stack for an item	پشته را برا ذخیره آدرس محل فراخوانی آماده میکنیم ←
sw x10, 0(sp) // save x10 to stack	مقدار n-1 را در پشته ذخیره ، تا بتوانیم مقدار Fib(n-2) را محاسبه کرد ←
jal x1, Int fib //run Int fib(n-1)	دوباره به برچسب Int fib رفته تا Fib(n-1) محاسبه شود و بازگردیم ←
lw x10, 0(sp) // load n-1	پس از محاسبه Fib(n-1)، n-1 را از پشته بازیابی و در x10 قرار میدهیم ←
addi sp, sp, 4 //adjust sp to pop 2 item	پشته را آزاد کرده ←
addi x10, x10, -1 //n=n-2	دوباره n را یک واحد کم تا Fib(n-2) محاسبه شود ←
jal x1, Int fib //run fib(n-2)	باز به برچسب Int fib رفته تا مقدار Fib را برای n-2 محاسبه کنیم ←

lw x1, 0(sp) // load return address addi sp, sp, 4 //adjust sp to pop 2 item jalr x0, x1, 0 // jump back to caller End: //jump to Int fib Jal x1, Int fib	پس از محاسبه و جمع fib(n-1)با fib(n-2) آدرس محل فراخوانی را بازیابی میکنیم ← پشته را آزاد کرده ← به محل فراخوانی برمیگردیم ← به برچسب Int fib رفته تا مقدار Fib(n) محاسبه شود ←

کد اجرایی R-fib

```
addi x11, x0, 0
addi x12, x0, 0
addi x5, x0, 1
addi x10, x0,7
addi sp, sp, 400
beq x0, x0, End
Int fib:
    addi sp, sp, -4
    sw x1, 0(sp)
    blt x5, x10, L
    beq x10, x5, 24
    addi x12, x0, 0
    lw x1, 0(sp)
    addi sp, sp, 4
    add x11, x11, x12
    jalr x0, x1, 0

    addi x12, x0, 1
    lw x1, 0(sp)
    addi sp, sp, 4
    add x11, x11, x12
    jalr x0, x1, 0
L:

    addi x10, x10, -1
    addi sp, sp, -4
    sw x10, 0(sp)
```

```
    jal x1, Int fib
    lw x10, 0(sp)
    addi sp, sp, 4

    addi x10, x10, -1
    jal x1, Int fib


    lw x1, 0(sp)
    addi sp, sp, 4
    jalr x0, x1, 0

End:
    Jal x1, Int fib
```

Convert to C

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 32 بیتی باشد که در رجیستر x10 گذاشته می‌شود و ورودی دیگر، عدد 32 بیتی دیگری باشد که نشانی شروع آرایه‌ای 32 عنصری از کاراکترها باشد و در رجیستر x11 گذاشته می‌شود.

جواب:

الف)

ابتدا اعداد را از مبنای 16 به مبنای 2 میبریم

01F00393
0000 0001 1111 0000 0000 0011 1001 0011
00755E33
0000 0000 0111 0101 0101 1110 0011 0011
001E7E13
0000 0000 0001 1110 0111 1110 0001 0011
01C580A3
0000 0001 1100 0101 1000 0000 1010 0011
00158593
0000 0000 0001 0101 1000 0101 1001 0011
FFF38393
1111 1111 1111 0011 1000 0011 1001 0011

FE03D6E7

1111 1110 0000 0011 1101 0110 1110 0111

00008067

0000 0000 0000 0000 1000 0000 0110 0111

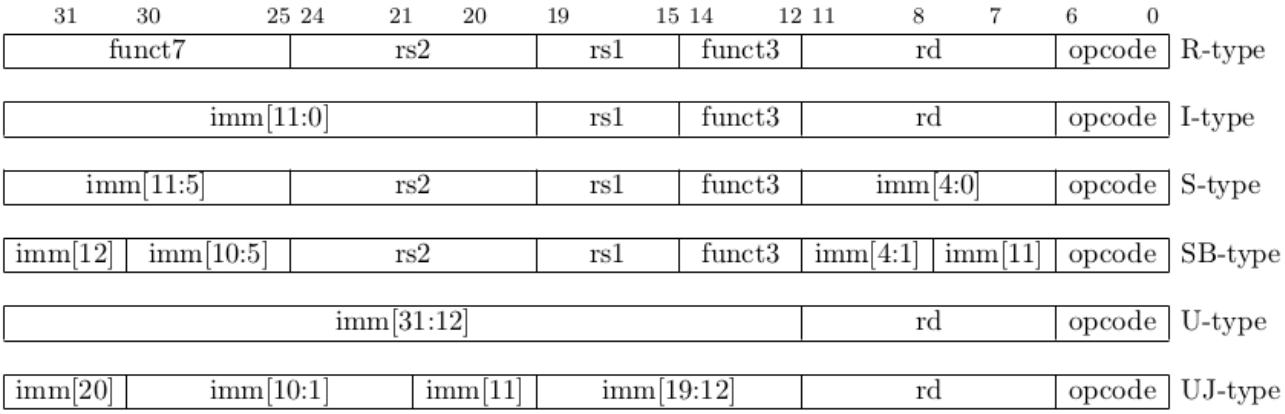

Figure 2.3: RISC-V base instruction formats showing immediate variants.

حال با توجه به جدول فوق 7 بیت سمت راست هر دستور نمایش دهنده opcodeبوده و با توجه به آن قالب دستورات را پیدا میکنیم

دستور 1:

0000 0001 1111 0000 0000 0011 1001 0011 opcode=0010011→I-type

پس بیت های 12تا 15 نمایش دهنده funct3 اند پس funct3=000 پس با توجه به جدول، دستور addi است.

پس بیت های 7-11 بیانگرrdو 15-20 برای rs1 و 20-31 برای imm است پس

rd=00111=7, rs1=00000=0, imm=0000 0001 1111=31

پس دستور آن برابر addi x7, x0, 31 است

دستور2:

0000 0000 0111 0101 0101 1110 0011 0011 →opcode=0110011→R-type

rd=11100 =28, funct3=101, rs1=01010=10, rs2=00111=7, funct7=0000000

پس با توجه به جدول و مقادیر فوق دستور متناظر آن برابر است با srl x28, x10, x7

دستور3:

0000 0000 0001 1110 0111 1110 0001 0011 →opcode= 0010011→I-type

rd=11100=28, funct3=111=7, rs1=11100=28, imm=000000000001=1

پس با توجه به جدول دستور متناظر آن برابر است با : andi x28, x28, 1

دستور 4:

0000 0001 1100 0101 1000 0000 1010 0011 →opcode= 0100011→S-type

→imm=000000000001=1, rs2=11100=28, rs1=01011=11, funct3= 000

پس دستور متناظر با آن برابر است با :sb x28, 1(x11)

دستور5:

0000 0000 0001 0101 1000 0101 1001 0011 →opcode=0010011→I-type
imm=000000000001=1,rs1=01011=11, funct3=000, rd=01011=11

پس دستور متناظر آن برابر است با: addi x11, x11, 1

دستور 6:

1111 1111 1111 0011 1000 0011 1001 0011 →opcode=0010011→I-type
imm=-1, rs1=00111, funct3=000, rd=00111

پس دستور متناظر آن با توجه به جدول برابر است با : addi x7, x7,-1

دستور 7:

1111 1110 0000 0011 1101 0110 1110 0111 → opcode= 1100111→SB- type

imm=1111111101100=-20→ سمت راست ترین بیت در این قالب همیشه صفرکه در دستور نوشته نمی شود است, rs2=00000, rs1=00111, funct3=101

پس دستور متناظر آن با توجه به جدول برابر است با : bge x7, x0,-20

دستور 8:

0000 0000 0000 0000 1000 0000 0110 0111 → opcode= 1100111→I- type
imm=000000000000=0, rs1=00001=1, rd=00000=0, funct3=000

پس دستور متناظر آن با توجه به جدول برابر است با : jalr x0,0(x1)

برنامه ماشین RISC-V	برنامه نمادین RISC-V
0x01F00393	addi x7, x0, 31 //x7=0+31=31
0x00755E33	srl x28, x10, x7 //shift x10 |x7|=31 bit to right and store it to x28
0x001E7E13	andi x28, x28, 1 //store and x28, 1 to x28
0x01C580A3	sb x28, 1(x11) //store a byte from x28 to memmory[x11+1]
0x00158593	addi x11, x11, 1 // x11=x11+1
0xFFF38393	addi x7, x7, -1 //x7=x7-1
0xFE03D6E7	bge x7, x0, -20 //if x7>=0 go to PC+(-20) //32 times repeat
0x00008067	jalr x0, 0(x1) //return to x1

ب)

برنامه C	برنامه نمادین RISC-V
Void fun(int x10 , char A[]){	
int x7=31, x28;	addi x7, x0, 31 //x7=0+31=31 →
char A[32];	
int x11= &A;	
for (x7; x7>=31; x7--) {	

x28= x10>>x7;
x28=x28&1;
memory[x11+1]=char(x28);
x11++;
	srl x28, x10, x7 //shift |x7| right x10 and store it to x28 →
	andi x28, x28, 1 //store and x28, 1 to x28
	sb x28, 1(x11) //store a byte from x11 to memory[x28+1]
	addi x11, x11, 1 // x11=x11+1
	addi x7, x7, -1 //x7=x7-11
	bge x7, x0, -20 //if x7>=0 go to PC+(-20)

}	
}	jalr x0, 0(x1) //return to x1

** تذکر: چون ذخیره سازی در آرایه از اندیس 1 شروع شده نه از 0 و 32 بار حلقه تکرار شده پس در آخرین تکرار ذخیره سازی آخرین عمل ذخیره سازی از آرایه A سرریز خواهد کرد.**

پ)

در ابتدا x7 برابر 31 میشود سپس x11 آدرس شروع آراییه 32 کارکتری را در خود ذخیره میکند.

سپس درون حلقه در ، به ازای هر تکرار ، x7 از 31 تا 0 یعنی 32 بار، حلقه تکرار میشود . در هر تکرار حلقه ابتدا x10 که عددی 32 بیتی بوده به مقدار x7 که در ابتدا 31 و در آخرین تکرار حلقه 0 است به راست منتقل شده که یعنی در تکرار اول ،عدد حاصل شامل 31 بیت صفر از سمت چپ و یک بیت سمت راست0 یا 1 که با ارزش ترین بیت عدد x10 بوده تشکیل میشود و در تکرار آخر نیز همان x10 بی تغییر باقی میماند و در x28 ذخیره میشود . پس از انتقلال ؛ 8 بیت سمت راست عدد x28با 1 and منطقی میشود که خروجی آن همیشه یا 1 است یا صفر زیرا 1 به صورت 31 صفر در سمت چپ و یک صفر در سمت راست ذخیره میشود. سپس 8 بیت کم ارزش x28که برابر یک کاراکتر است در حافظه ذخیره شده و x11 که نشانی شروع آرایه در حافظه است یک واحد افزایش و حلقه دوباره تکرار میشود.

در نهایت در آرایه 32 عنصر که آدرس شروع آن در x11 قرار داده شده بود ذخیره میشود که هر کاراکتر آن برابر 00000000 یا 00000001 بوده که 0 یا 1 سمت راست هر کاراکتر آن به ترتیب وارون عدد ذخیره شده در رجیستر x10 بوده برای مثال اگر x10 را 2 بیتی و برابر 10 در نظر بگیریم در آرایه 2 عضوی درون حافظه 00000000 به عنوان کاراکتر اول و 0000001 به عنوان کارکتر دوم آرایه ذخیره خواهد شد.