Homework 3

Benny Chen

September 28, 2022

Question 1

Translate function foo() in the following C code to RISC-V assembly code. Assume function bar() has already been implemented. The constraints/tips are:

1. Allocate register s1 to sum, and register s2 to i.
2. There are no load or store instructions in the loop. If we want to preserve values across function calls, place the value in a saved register before the loop. For example, we keep variable i in register s2.
3. Identify the registers that are changed in function foo() but should be preserved. Note that the callee, bar(), may change any temporary and argument registers
4. Save registers at the beginning of the function and restore them before the exit.

Your code should follow the flow of the C code. Write concise comments. Clearly mark instructions for saving registers, loop, function calls, restoring register, etc.

```
// prototype of bar
// the first argument is an address of an integer
int  bar(int a[], int i);

int foo(int d[], int n)
{
    int sum = 0;
    for (int i = 0; i < n; i += 1) {
        sum += bar(&d[i], n - i);    // &d[i] means d[i]'s address
    }
    return sum;
}
```

Answer:

```
 foo:
     addi    sp,sp,-20 #Allocate space
     sw      s1,0(sp)
     sw      s2,4(sp)
     sw      s3,8(sp)
     sw      s4,12(sp)
     sw      ra,16(sp)

     addi    s1,s1,0 #s1 = sum = 0
     addi    s2,s2,0 #s2 = i = 0
     addi    s3,s3,0 #s3 = d address = ?
     addi    s4,s4,100 #s4 = n = ?

loop:
     slli    a0,s2,2 #offset of i
     add     a0,a0,s3 #&d[i]
     sub     a1,s4,s2 #n-i
     jal     ra,bar #bar(&d[i],n-i)

     add     s1,s1,a0 #sum += output of bar(&d[i],n-i)

     addi    s2,s2,1 #i+=1
     blt     s2,s4,loop #if i < n

return:
     addi    a0,s1,0 #foo returns sum so into a0

     #restore all
     lw      s1,0(sp)
     lw      s2,4(sp)
     lw      s3,8(sp)
     lw      s4,12(sp)
     lw      ra,16(sp)

     addi    sp,sp,20

     jr ra
```

Question 2

Translate function msort() in the following C code to RISC-V assembly code. Assume merge() and copy() are already implemented. The array passed to msort() has at most 256 elements. Your code should follow the flow of the C code. Write concise comments. Clearly mark instructions for saving registers, function calls, restoring register, and so on. To make the code easier to read, we change sp twice at the beginning of the function: once for saving registers and once for allocating memory for array c. The function should have only one exit. There is only one return instruction. Another reminder: callees may change any temporary and argument registers.

```
void merge(int c[], int d1[], int n1, int d2[], int n2);
void copy(int d[], int c[], int n);

void  msort(int d[], int n)
{
    int c[256];

    if (n <= 1)
        return;

    int n1 = n / 2;

    msort(d, n1);
    msort(&d[n1], n - n1);    // &d[n1] means the address of d[n1]
    merge(c, d, n1, &d[n1], n - n1);
    copy(d, c, n);
}
```

Answer:

```
# merge inputs in (c[], d1[], n1, d2[], n2)
# copy inputs in (d[], c[], n)

# Inputs in d[] and int n
msort:
        addi    sp,sp,-1036 #allocate space for variables

        sw      ra,1032(sp)
        sw      s2,1028(sp)
        sw      s1,1024(sp)

        addi    s1,a0,0 #Save D[] to s1
        addi    s2,a1,0 #Save n to s2

        addi    t0,t0,2 # t0 = 2 for if statement
        blt             a1,t0,exit # return if n < 2

        srai    t1,s2,1 # n1 = n/2
        addi    a1,t1,0 # a1 = n1
        jal     ra,msort  # call msort(d,n1)

        slli    t3,t1,2 #t3 = n1 * 4
```

```
        add     a0,t3,s1 #n1 + &d = &d[n1] = a0
        sub     a1,s2,t1 #n-n1 = a1
        jal     ra,msort  # call msort(&d[n1], n - n1)

        addi    a0,sp,0 #c
        addi    a1,s1,0 #d
        addi    a2,t1,0 #n1
        add     a3,t3,s1 #n1 + &d = &d[n1] = a3
        sub     a4,s2,t1 #n-n1 = a4
        jal     ra,merge #merge(c, d, n1, &d[n1], n - n1)

        addi    a0,s1,0 #d
        addi    a1,sp,0 #c
        addi    a2,s2,0 #n
        jal     ra,copy #copy(d, c, n)

exit:
        #restore registers
        lw      s1,1024(sp)
        lw      s2,1028(sp)
        lw      ra,1032(sp)

        addi    sp,sp,1036

        jr      ra
```

Question 3

Find the machine code for the following instruction. Assume all instructions are labeled sequentially, for example, I1, I2, I3, . . . , I150.

```
...
I10 :        BGE   x10, x20, I100
I11 :        BEQ   x10, x0, I1
...
I140:        JAL   x0, I100
```

CORE INSTRUCTION FORMATS

	31–27	26–25	24–20	19–15	14–12	11–7	6–0
R	funct7		rs2	rs1	funct3	rd	Opcode
I	imm[11:0]			rs1	funct3	rd	Opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

I10: BGE x10,x20,I100

Opcode: 1100011
funct3: 101
Type: SB
rs1: 01010
rs2: 10100
work: (100 - 10) * 4 = 360
immediate: 0000101101000
Machine code: 0001011 10100 01010 101 01000 1100011
Hex: 0x17455463

im[12][10:5]	rs2	rs1	funct3	im[4:1][11]	Opcode
0001011	10100	01010	101	01000	1100011

I11: BEQ x10,x0,I1

Opcode: 1100011
funct3: 000
Type: SB
rs1: 01010
rs2: 00000
work: (1 - 11) * 4 = -40
immediate: 1111111011000
Machine code: 1111110 00000 01010 000 11001 1100011
Hex: 0xFC050CE3

im[12][10:5]	rs2	rs1	funct3	im[4:1][11]	Opcode
1111110	00000	01010	000	11001	1100011

I140: JAL x0,I100

Opcode: 1101111
Type: UJ
rd: 00000
work: (100 - 140) * 4 = -160
immediate: 111111111111101100000
Machine code: 11110110000111111111 00000 1101111
Hex: 0xF61FF06F

imm[20][10:1][11][19:12]	rd	Opcode
11110110000111111111	00000	1101111

Question 4

Decode the following instructions in machine code. Find the offset in decimal and then the target address in hexadecimal. The hexadecimal number before the colon is the instruction's address.

`0x0400366C:` `0xDB5A04E3`

`0x04208888:` `0xFA9FF0EF`

CORE INSTRUCTION FORMATS

	31 27 26 25	24 20	19 15	14 12	11 7	6 0
R	funct7	rs2	rs1	funct3	rd	Opcode
I	imm[11:0]		rs1	funct3	rd	Opcode
S	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]				rd	opcode
UJ	imm[20|10:1|11|19:12]				rd	opcode

0x0400366C: 0xDB5A04E3

Instruction in Binary: 1101101 10101 10100 000 01001 1100011
Opcode: 1100011
funct3: 000
Type: SB
rs1: 10100
rs2: 10101
immediate work: [1][1][101101][0100][0]
immediate: 1110110101000
Decimal: -600
Target address: 0x0400366C - 0x258 = 0x4003414
Instruction: BEQ x20,x21,0x4003414

0x04208888: 0xFA9FF0EF

Instruction in Binary: 11111010100111111111 00001 1101111
Opcode: 1101111
Type: UJ
rd: 00001
immediate work: [1][11111111][1][1111010100][0]
immediate: 111111111111110101000
Decimal: -88
Target address: 0x04208888 - 0x58 = 0x4208830
Instruction: JAL x1,0x4208830