Recitation 10/30

Instruction Formats and RISC-V Encoding - RType

Format - instruction destination, source1, source2

Encoding

funct7	rs2	rs1	funct3	rd	opcode	R-type

Purpose - change value in destination to a manipulation of values in source1 and source2

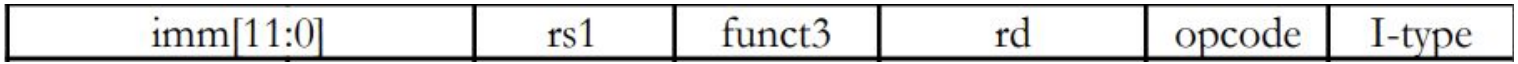
Example

add rd, rs1, rs2

Instruction Formats and RISC-V Encoding - IType

Format - instruction destination, source1, immediate

Encoding



Purpose - change value in destination to a manipulation of values in source1 and immediate

Example

addi rd, rs1, imm12

Purpose - set value in destination to value at memory location stored in source1 at some offset specified by immediate

Example

lb rd, imm12(rs1)

Instruction Formats and RISC-V Encoding - SType

Format - instruction register to be stored, offset(register with memory address)

Encoding

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type

Example

sw rs2, imm12(rs1)

Instruction Formats and RISC-V Encoding - BType

Format - instruction source1, source2, PC

Encoding

imm[12,10:5]	rs2	rs1	funct3	imm[4:1,11]	opcode	B-type

Purpose - compare values at register 1 and 2 and branch to location in counter if condition satisfied

Example

beq rs1, rs2, targ12

Instruction Formats and RISC-V Encoding - UType

Format - instruction destination, immediate

Encoding

imm[31:12]	rd	opcode	U-type

Example

lui rd, imm20

Instruction Formats and RISC-V Encoding - JType

Format - instruction destination, PC

Encoding

Purpose - store location of next instruction in destination, and change program counter

Example

jal rd, targ20

Practice Questions

What does the RISC-V code do?

Register s0 holds address of element at index 0 in arr, where *arr = {1, 2, 3, 4, 5, 6, 0}

a)
```
lw t0, 12(s0)
```
b)
```
sw t0, 16(s0)
```
c)
```
slli t1, t0, 2
add t2, s0, t1
lw t3, 0(t2)
addi t3, t3, 1
sw t3, 0(t2)
```
d)
```
 lw t0, 0(s0)
xori t0, t0, 0xFFF
addi t0, t0, 1
```

What does the RISC-V code do?

Register s0 holds address of element at index 0 in arr, where *arr = {1, 2, 3, 4, 5, 6, 0}

a) lw t0, 12(s0) - load arr[3] into t0
b) sw t0, 16(s0) - store arr[4] into
c) slli t1, t0, 2
   add t2, s0, t1
   lw t3, 0(t2)
   addi t3, t3, 1
   sw t3, 0(t2)
d) lw t0, 0(s0)
   xori t0, t0, 0xFFF
   addi t0, t0, 1

RISC-V branching commands to jump to LABEL

a) s0 < s1

b) s0 ≠ s1

c) s0 ≤ s1

d) s0 > s1

RISC-V branching commands to jump to LABEL

a) s0 < s1

   blt s0, s1, LABEL

b) s0 ≠ s1

   bne s0, s1, LABEL

c) s0 ≤ s1

   bge s1, s0, LABEL

d) s0 > s1

   blt s1, s0, LABEL

Translate to RISC-V

```
// s0 -> a, s1 -> b

// s2 -> c, s3 -> z

int a = 4, b = 5, c = 6, z;

z = a + b + c + 10;
```

Translate to RISC-V

```
// s0 -> a, s1 -> b
// s2 -> c, s3 -> z
int a = 4, b = 5, c = 6, z;
z = a + b + c + 10;
```

```
addi s0, x0, 4
addi s1, x0, 5
addi s2, x0, 6
add s3, s0, s1
add s3, s3, s2
addi s3, s3, 10
```

Translate to RISC-V

```
// s0 -> int * p = intArr;

// s1 -> a;

*p = 0;

int a = 2;

p[1] = p[a] = a;
```

Translate to RISC-V

```
// s0 -> int * p = intArr;
// s1 -> a;
*p = 0;
int a = 2;
p[1] = p[a] = a;
```

```
sw x0, 0(s0)
addi s1, x0, 2
sw s1, 4(s0)
slli t0, s1, 2
add t0, t0, s0
sw s1, 0(t0)
```

Translate to RISC-V

```
// s0 -> a, s1 -> b

int a = 5, b = 10;

if(a + a == b) {

        a = 0;

} else {

        b = a - 1;

}
```

Translate to RISC-V

```
// s0 -> a, s1 -> b
int a = 5, b = 10;
if(a + a == b) {
        a = 0;
} else {
        b = a - 1;
}
```

```
addi s0, x0, 5
addi s1, x0, 10
add t0, s0, s0
bne t0, s1, else
 xor s0, x0, x0
jal x0, exit
else:
        addi s1, s0, -1
exit:
```

Translate to C

```
        addi s0, x0, 0
        addi s1, x0, 1
        addi t0, x0, 30
loop:
        beq s0, t0, exit
        add s1, s1, s1
        addi s0, s0, 1
        jal x0, loop
exit:
```

Translate to C

```
// computes s1 = 2^30

// assume int s1, s0; was declared above s1 = 1;

for(s0 = 0; s0 != 30; s0++) {

        s1 *= 2;

}
```

```
        addi s0, x0, 0
        addi s1, x0, 1
        addi t0, x0, 30
loop:
        beq s0, t0, exit
        add s1, s1, s1
        addi s0, s0, 1
        jal x0, loop
exit:
```

Translate to RISC-V

```
// s0 -> n, s1 -> sum
// assume n > 0 to start
for(int sum = 0; n > 0; n--) {
        sum += n;
}
```

Translate to RISC-V

```
// s0 -> n, s1 -> sum
// assume n > 0 to start
for(int sum = 0; n > 0; n--) {
        sum += n;
}
```

```
        addi s1, x0, 0
loop:
        beq s0, x0, exit
        add s1, s1, s0
        addi s0, s0, -1
        jal x0, loop
exit:
```

Decoding an Instruction

Without looking at the instruction sheet what can we tell about these instructions?

```
iiii iiit tttt ssss s000 iiii i110 0011
```

```
0000 000t tttt ssss s111 dddd d011 0011
```

How do we write an assembly program?

- Think about how to solve the problem
- Translate into pseudo-code or a language of your choice
- Write down what registers you will use for what tasks

Try it out!

Let’s write factorial