



**Control-Flow Instructions**

- The first two instructions use a different addressing mode called pc-relative addressing at the resolution of 12 bit.
- **Branch on equal** sets the `pc` to `pc + imm` if the content of `$rs1` matches `$rs2`.
- **Jump and link** is used for procedure calls and stores the return address (address of next instruction) in `$rd`.
- **Jump and link register** is similar to `jal`, except that it uses register relative addressing in order to jump a little further.

beq

\$rs1

\$rs2

imm

jal

\$rd

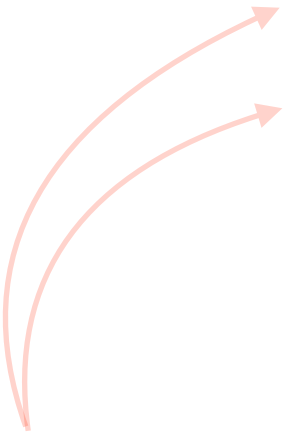
imm

jalr

\$rd

\$rs1

imm



# Link Register

```
uint64_t increment(uint64_t inc) {  
    return inc + 1;  
}
```

```
uint64_t main() {  
    uint64_t a;
```

```
    a = 0
```

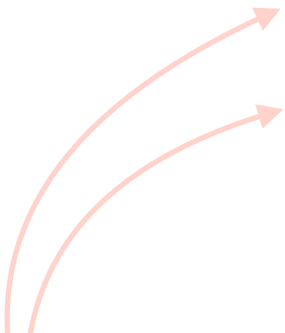
```
    a = increment(a);
```

```
    return a;
```

```
}
```

The next **instruction**  
that is linked is the  
assignment into a!

# Control-Flow Instructions



beq	<code>\$rs1</code>	<code>\$rs2</code>	<code>imm</code>
jal	<code>\$rd</code>	<code>imm</code>	
jalr	<code>\$rd</code>	<code>\$rs1</code>	<code>imm</code>

- The first two instructions use a different addressing mode called pc-relative addressing at the resolution of 12 bit.
- **Branch on equal** sets the pc to  $pc + imm$  if the content of `$rs1` matches `$rs2`.
- **Jump and link** is used for procedure calls and stores the return address (address of next instruction) in `$rd`.
- **Jump and link register** is similar to `jal`, except that it uses register relative addressing in order to jump a little further.