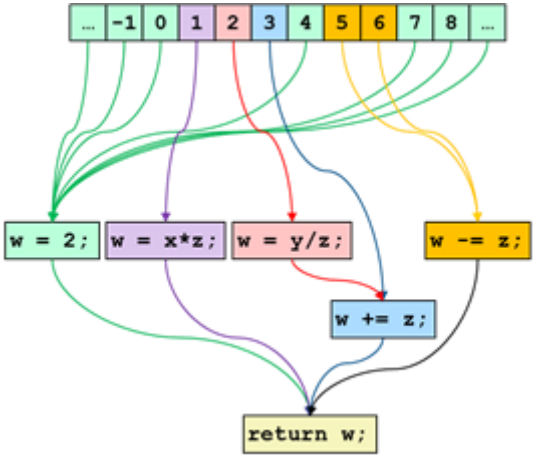
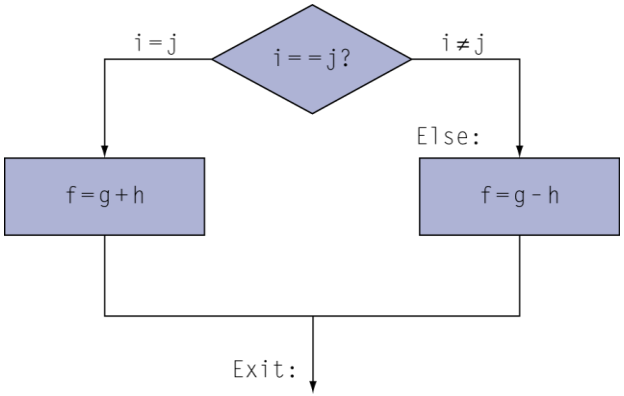
# The HW/SW Interface

# RISC-V
# Control Flow



# Module Outline

- **Altering Control Flow**
  - **If-then-else Constructs**
  - **Loop Constructs**
    - do { … } while (…)
    - while (…) do { … }
    - for (…) do { … }
- **Module Summary**




# Altering Control Flow

# Altering Control Flow

- Higher-level programming languages offer control-flow altering constructs

```
int foo(int x, int y)
{
  int res = 0;

  if (x > y) x = x-y;

  while (x > 0) {
    res = res + y;
    x--;
  }

  return res;
}
```

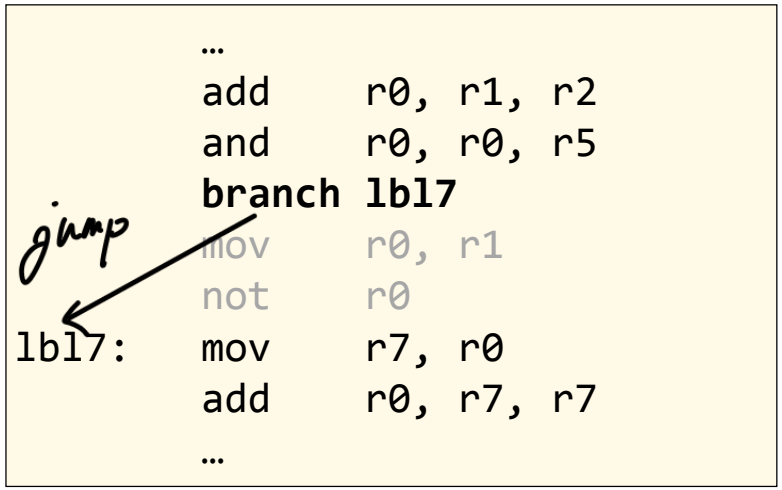
- How can we achieve that with assembly code?

# Branch Operations

- Processor ISAs offer branch operations to alter the sequential control flow
  - generic form

```
branch <label>
```

  - instructs processor to continue execution at <label>
    - same as goto in higher-level programming languages
  - example
    - branch <label> implemented as
      PC = &label
  - branch is always executed
    = **unconditional branch**



# Branch Operations

- Processor ISAs offer branch operations to alter the sequential control flow
  - generic form

```
branch <label>
```

  - instructs processor to continue execution at <label>
    - same as goto in higher-level programming languages
  - branch <label> implemented as
    PC = &label
  - **unconditional branch**
    - branch is always executed
  - differences by architectures
    - Intel:

```
jmp    <label>
```

    - RISC-V:

```
branch <label>
```

```
        …
        add     r0, r1, r2
        and     r0, r0, r5
        branch lbl7
        mov     r0, r1
        not     r0
lbl7:   mov     r7, r0
        add     r0, r7, r7
        …
```

# Branch Operations

- **Conditional branches**
  - *conditionally* alter control flow
  - generic form

```
branch <condition>, <label>
```

  - instructs processor to continue execution at <label> if <condition> is true
    - if (condition) goto label
  - different ways to implement conditional branches

```
if (operand1 <cond> operand2) goto <label>
```

    - Intel:

```
cmp     operand2, operand1
j<cond> <label>
```

check flag

    - RISC-V:

```
b<cond> operand1, operand2, <label>
```

# RISC-V Branch Operations

- **Core branch operations**

| Conditional branch | Description | Data type |
|---|---|---|
| beq == | branch if equal | d/c |
| bne != | branch if not equal | d/c |
| bge >= | branch if greater than or equal | signed |
| blt < | branch if less than | signed |
| bgeu >= | branch if greater than or equal | unsigned |
| bltu < | branch if less than | unsigned |

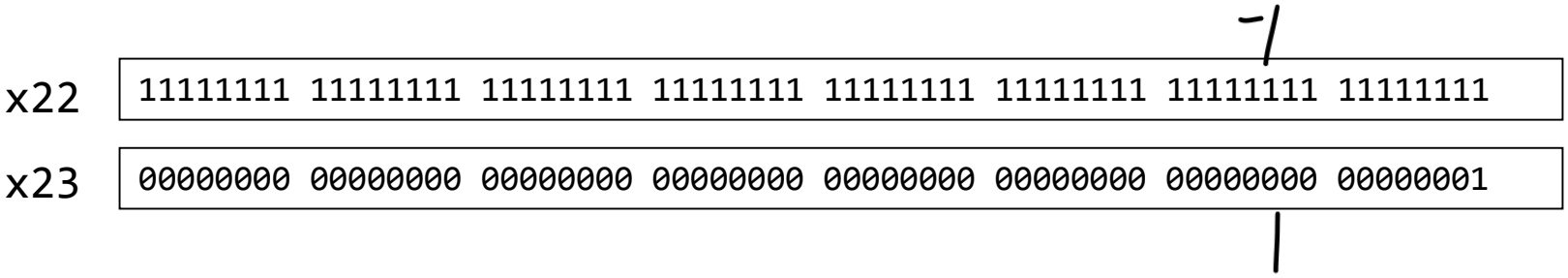
- no unconditional branch?!

  there is: `beq x0, x0, <label>` ! (always true)

# Signed vs. Unsigned Comparison

- Signed comparison: `blt`, `bge`
- Unsigned comparison: `bltu`, `bgeu`

- Example




```
blt   x22, x23, Exit
```

**= Go to Exit if -1 < 1**

```
bltu  x22, x23, Exit
```

**= Go to Exit if $2^{64}$-1 < 1**

# RISC-V Branch Operations

- **Pseudoinstructions** (pseudo implement with 6 core instruction)

| Conditional branch | Implemented as | Description |
|---|---|---|
| `bgt rs, rt, <lbl>` | `blt rt, rs, <lbl>` | branch if >, signed |
| `ble rs, rt, <lbl>` | `bge rt, rs, <lbl>` | branch if <=, signed |
| `bgtu rs, rt, <lbl>` | `bltu rt, rs, <lbl>` | branch if >, unsigned |
| `bleu rs, rt, <lbl>` | `bgeu rt, rs, <lbl>` | branch if <=, unsigned |
| `beqz rs, <lbl>` | `beq rs, x0, <lbl>` | branch if rs == 0 |
| `bnez rs, <lbl>` | `bne rs, x0, <lbl>` | branch if rs != 0 |
| `blez rs, <lbl>` | `bge x0, rs, <lbl>` | branch if <= 0 |
| `bgez rs, <lbl>` | `bge rs, x0, <lbl>` | branch if >= 0 |
| `bltz rs, <lbl>` | `blt rs, x0, <lbl>` | branch if < 0 |
| `bgtz rs, <lbl>` | `blt x0, rs, <lbl>` | branch if > 0 |

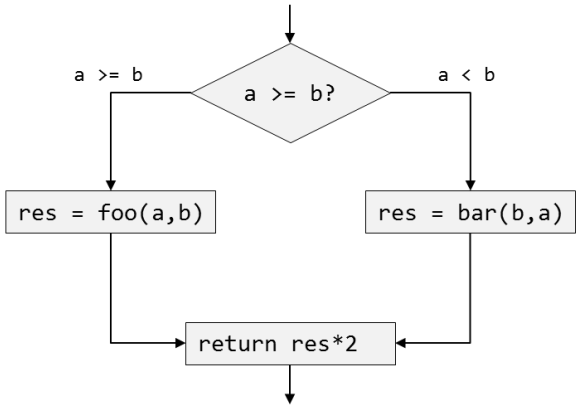
# Target Addressing

- Target addresses are always aligned to 2 bytes (i.e., even addresses)
  - Some of instructions can be encoded with 16 bits (with C extension)
  - PC-relative

2 bytes or 4 bytes

- Branch addressing
  - Most branch targets are near branch: forward or backward
  - Target address = PC + SignExt(12-bit immediate value << 1)

- Assembler computes offset to label




# Altering Control Flow

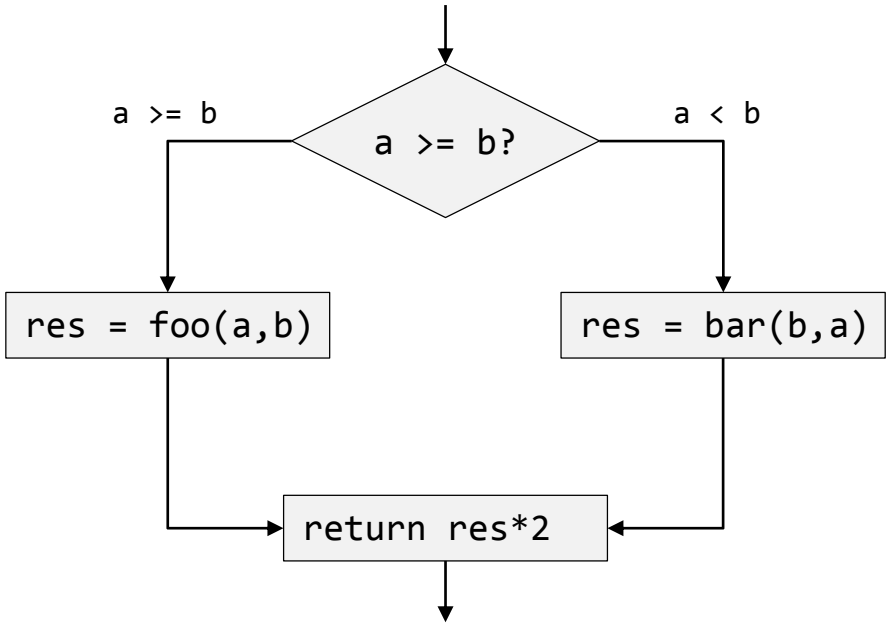
# If-then-else Constructs

# Compiling If Statements

**C code:**

```
int cf_if(long a, long b) {
  int res;

  if (a >= b) res = foo(a, b);
  else res = bar(b, a);

  return res*2;
}
```

**Control flow:**

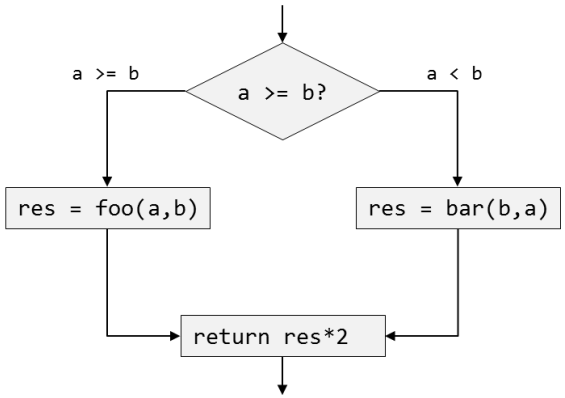


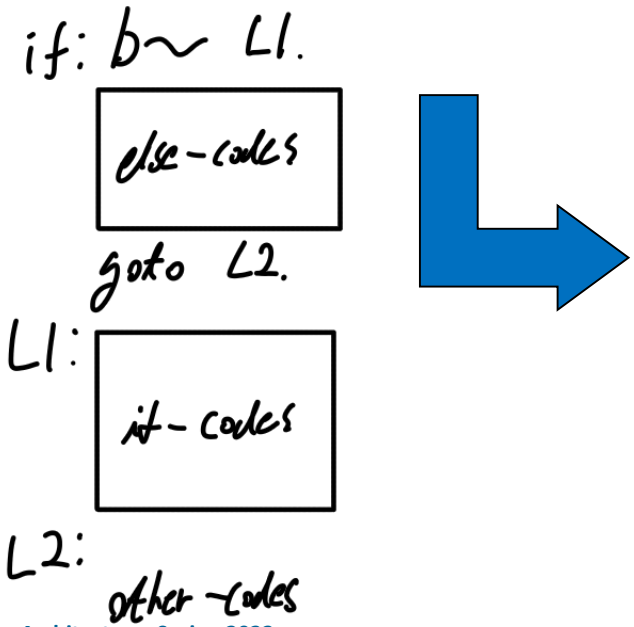
# Compiling If Statements

**C code:**

```
int cf_if(long a, long b) {
  int res;

  if (a >= b) res = foo(a, b);
  else res = bar(b, a);

  return res*2;
}
```

**Control flow:**




**C code (goto version):**

```
  if (a < b) goto L_false;
  res = foo(a, b)
  goto Exit;

L_false:
  res = bar(b, a)

Exit:
  return res*2;
```

# Compiling If Statements

**C code:**

```
int cf_if(long a, long b) {
  int res;

  if (a >= b) res = foo(a, b);
  else res = bar(b, a);

  return res*2;
}
```

**C code (goto version):**

```
  if (a < b) goto L_false;
  res = foo(a, b)
  goto Exit;

L_false:
  res = bar(b, a)

Exit:
  return res*2;
```

**Direct translation to RISC-V code:**

```
cf_if:
  addi   sp,sp,-16
  sw     ra,12(sp)

  mv     a5,a1
  blt    a0,a1,.L_false

  call   foo
  j      .L_exit

.L_false:
  mv     a1,a0
  mv     a0,a5
  call   bar

.L_exit:
  slli   a0,a0,1

  lw     ra,12(sp)
  addi   sp,sp,16
  jr     ra
```

# Compiling If Statements

**C code:**

```
int cf_if(long a, long b) {
  int res;

  if (a >= b) res = foo(a, b);
  else res = bar(b, a);

  return res*2;
}
```

**C code (goto version):**

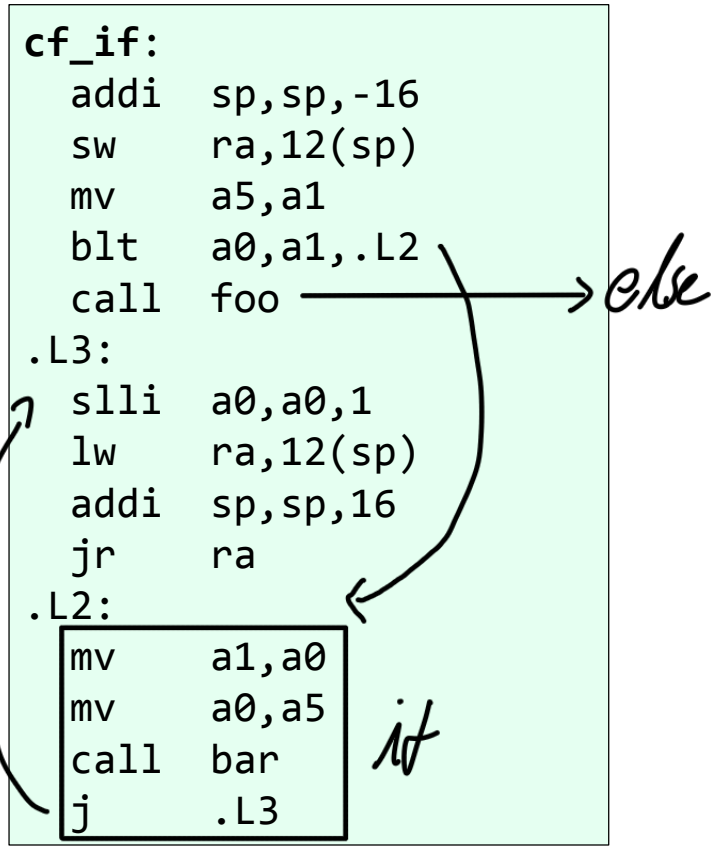
```
  if (a < b) goto L_false;
  res = foo(a, b)
  goto Exit;

L_false:
  res = bar(b, a)

Exit:
  return res*2;
```

**Direct translation to RISC-V code:**

```
cf_if:
  addi   sp,sp,-16
  sw     ra,12(sp)

  mv     a5,a1
  blt    a0,a1,.L_false

  call   foo
  j      .L_exit

.L_false:
  mv     a1,a0
  mv     a0,a5
  call   bar

.L_exit:
  slli   a0,a0,1

  lw     ra,12(sp)
  addi   sp,sp,16
  jr     ra
```

# Compiling If Statements

**Code generated by GCC 9.2**




- Observations
  - minimize control flow instructions
  - minimize instructions in general
  - aligns stack points at 16-byte boundaries
  - condition sometimes reversed
    - and fix jumps to if/else bodies

# Compiling If Statements

**RISC-V Code**

```
cf_if:
  addi   sp,sp,-16
  sw     ra,12(sp)
  mv     a5,a1
  blt    a0,a1,.L_false
  call   foo
  j      .L_exit

.L_false:
  mv     a1,a0
  mv     a0,a5
  call   bar

.L_exit:
  slli   a0,a0,1
  lw     ra,12(sp)
  addi   sp,sp,16
  jr     ra
```

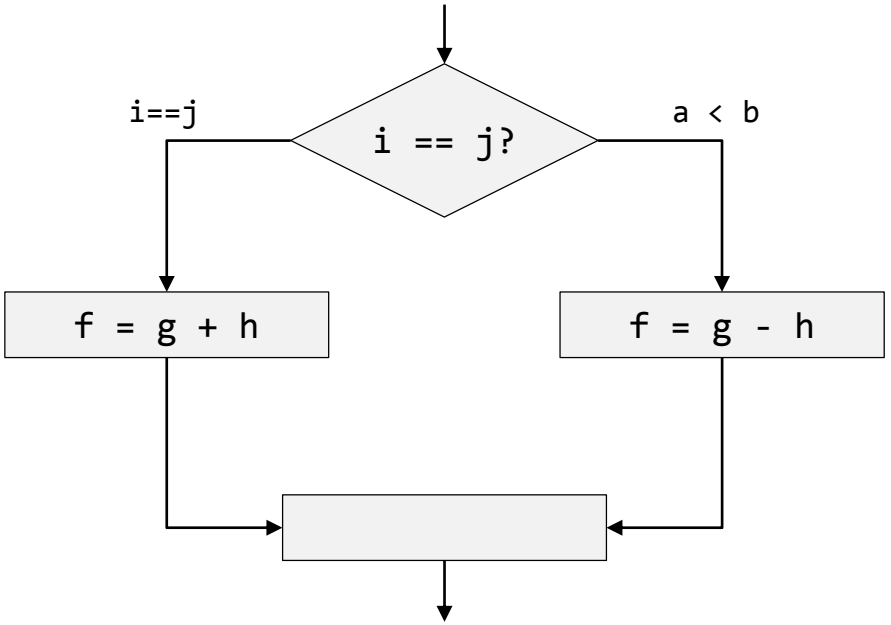
**x86 Code:**

```
cf_if:
    movq    %rdi, %r8
    subq    $8, %rsp
    movq    %rsi, %rdi
    cmpq    %rsi, %r8
    jl      .L_false
    movq    %r8, %rdi
    call    foo@PLT
    jmp     .L_exit

.L_false:
    movq    %r8, %rsi
    call    bar@PLT

.L_exit:
    addq    $8, %rsp
    addl    %eax, %eax
    ret
```

# Another Example

**C code:**

```
if (i == j)
  f = g + h;
else
  f = g – h;
```

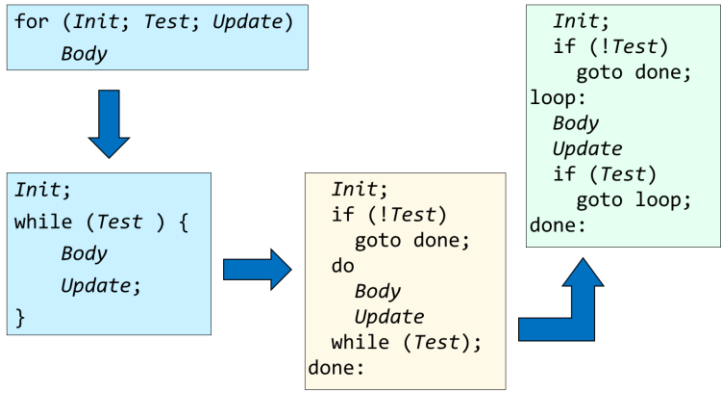



**C code (goto version):**

```
      if (i != j) goto L1;
      f = g + h;
      goto Exit;
L1:   f = g – h;
Exit:
```

**Compiled RISC-V code:**

```
// i in x22, j in x23
// f in x19, g in x20, h in x21

      bne  x22, x23, L1
      add  x19, x20, x21
      beq  x0, x0, Exit    // uncond.
L1:   sub  x19, x20, x21
Exit: ...
```




# Altering Control Flow

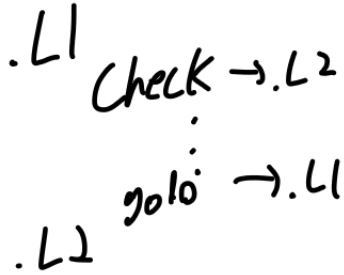
# Loop Constructs

# Loop Statements

- **Basic loop constructs**
  - do { body } while (cond);

```
do {
  i += 1;
} while (A[i] < i);
```

  - while (cond) { body }




```
while (A[i] == k) {
  i += 1;
};
```

  - for (init; cond; update) { body }

```
for (i=0; i<N; i++) {
  sum += A[i];
}
```

# General "Do-While" Translation

**C Code**

```
do
  Body
while (Test);
```

**Goto Version**

```
loop:
  Body
  if (Test)
    goto loop;
```

- Body:
  ```
  {
    Statement1;
    Statement2;
      …
    Statementn;
  }
  ```

- Test returns integer
  - = 0 interpreted as false
  - ≠ 0 interpreted as true

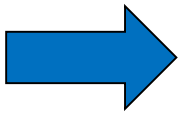
# General "While" Translation

**While version**

```
while (Test)
  Body
```

**Do-While Version**

```
  if (!Test)
    goto done;
  do
    Body
  while (Test);
done:
```



**Goto Version** normal while

```
  if (!Test)
    goto done;
loop:
  Body
  if (Test)
    goto loop;
done:
```

do-while

# Loop Statements

- **While as Do-While**
  - initial check
  - move condition to end

```
while (A[i] == k) {
  i += 1;
};
```

```
if (A[i] != k) goto Exit;
do {
  i += 1;
} while (A[i] == k);
Exit:
```

# “For” Loop → While → Do While → Goto

## For Version

```
for (Init; Test; Update)
    Body
```

## While Version

```
Init;
while (Test ) {
    Body
    Update;
}
```

## Do-While Version

```
  Init;
  if (!Test)
    goto done;
  do
    Body
    Update
  while (Test);
done:
```

## Goto Version

```
  Init;
  if (!Test)
    goto done;
loop:
  Body
  Update
  if (Test)
    goto loop;
done:
```

Basic of for

# "For" Loop Conversion Example

## C Code

```
#define WSIZE 8*sizeof(int)
int pcount_for(unsigned x) {
  int i;
  int result = 0;
  for (i = 0; i < WSIZE; i++) {
    unsigned mask = 1 << i;
    result += (x & mask) != 0;
  }
  return result;
}
```

- In this case, the initial test can be optimized away

## Goto Version

```
int pcount_for_gt(unsigned x) {
  int i;
  int result = 0;
  i = 0;                          Init
  if (!(i < WSIZE))               ! Test
    goto done;
 loop:                            Body
  {
    unsigned mask = 1 << i;
    result += (x & mask) != 0;
  }
  i++;                            Update
  if (i < WSIZE)                  Test
    goto loop;
 done:
  return result;
}
```

# Compiling Loop Statements

**C code:**

```
do {
  i += 1;
} while (A[i] == k);
```

**C code (goto version):**

```
Loop: i += 1;
      if (A[i] == k) goto Loop;
```

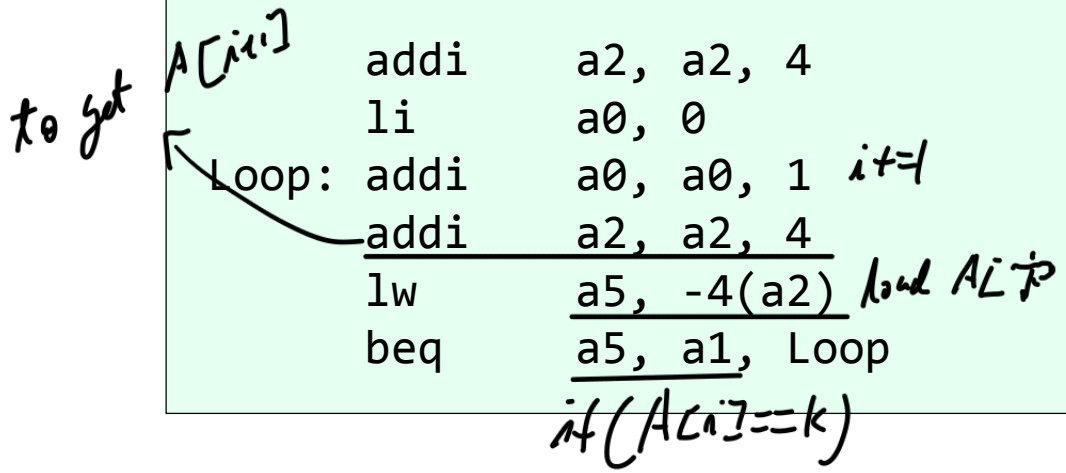
# Compiling Loop Statements

**C code (goto version):**

```
Loop: i += 1;
      if (A[i] == k) goto Loop;
```

**Compiled RISC-V code:**

```
// i in a0, k in a1, a5 = A[i]
// address of A[] in a2

      addi    a2, a2, 4
      li      a0, 0
Loop: addi    a0, a0, 1
      addi    a2, a2, 4
      lw      a5, -4(a2)
      beq     a5, a1, Loop
```



# Compiling Loop Statements

**C code:**

```
while (A[i] == k)
  i += 1;
```

**C code (goto version):**

```
Loop: if (A[i] != k) goto Exit;
      i += 1;
      goto Loop;
Exit:
```

# Compiling Loop Statements

**C code (goto version):**

```
Loop: if (A[i] != k) goto Exit;
      i += 1;
      goto Loop;
Exit:
```

**Compiled RISC-V code:**

```
 // i in x22, k in x24, A[i] in x9
 // address of A[] in x25

Loop: slli  x10, x22, 3
      add   x10, x10, x25
      ld    x9, 0(x10)
      bne   x9, x24, Exit
      addi  x22, x22, 1
      beq   x0, x0, Loop
 Exit: ...
```

get address of A[i]

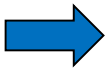
A[i] comp k

i = i+1

# While Loop Example (1)

```
long fact_while (long x)
{
    long result = 1;
    while (x > 1) {
        result *= x;
        x = x - 1;
    }
    return result;
}
```

```
long fact_while (long x)
{
   long result = 1;
Loop:
   if (x <= 1) goto Exit;
   result = result * x;
   x = x - 1;
   goto Loop;
Exit:
   return result;
}
```



**gcc with –Og option**

```
     # x is in a0

fact_while:
     addi    a5, a0, 0          # a5 = x (x)
     addi    a0, zero, 1        # a0 = 1 (result)
L2:
     addi    a4, zero, 1        # a4 = 1
     ble     a5, a4, L4         # if (x<=1) goto L4
     mul     a0, a0, a5         # result *= x
     addi    a5, a5, -1         # x = x - 1
     beq     zero, zero, L2     # goto L2
L4:
     ret
```

# While Loop Example (2)

```
long fact_while (long x)
{
    long result = 1;
    while (x > 1) {
        result *= x;
        x = x - 1;
    }
    return result;
}
```

```
long fact_while (long x)
{
    long result = 1;
Loop:
    if (x <= 1) goto Exit;
    result = result * x;
    x = x - 1;
    goto Loop;
Exit:
    return result;
}
```

while

**gcc with –O2 option**

```
    # x is in a0
                          a5=x , a4=1
fact_while2:
    addi    a5, a0, 0       # a5 = x (x)
    addi    a4, zero, 1     # a4 = 1 long result=1
    addi    a0, zero, 1     # a0 = 1 (result)
    ble     a5, a4, L4      # if (x<=1) goto L4
L3:
    mul     a0, a0, a5      # result *= x
    addi    a5, a5, -1      # x = x - 1
    bne     a5, a4, L3      # if (x!=1) goto L3
L4:
    ret
```

# For Loop Example

don't use i

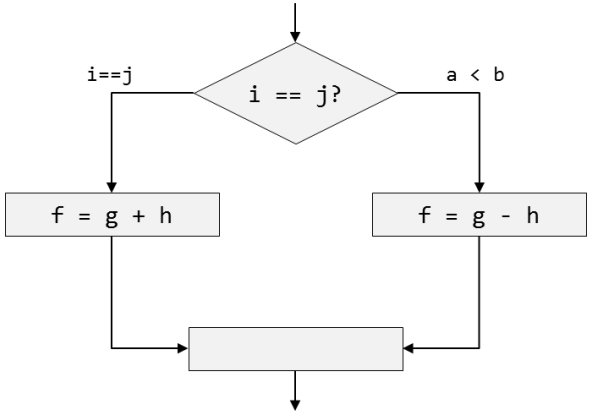
```
int sumarray(int *A, int N)
{
  int sum = 0;

  for (int i=0; i<N; i++) {
    sum += A[i];
  }

  return sum;
}
```

**gcc with –O1 option**

A = a5

```
    # A is in a0, N in a1

sumarray:
        blez    a1,.L4      # N == 0? goto L4
        mv      a5,a0       # a5 = copy &A[i(=0)]
        slli    a1,a1,2     # a1 = a1 * 4   N*4
        add     a3,a0,a1    # a3 = &A[N]
        li      a0,0        # sum = 0       compare with A[0]
.L3:
        lw      a4,0(a5)    # a4 = A[i]
        add     a0,a0,a4    # sum = sum + a4(=A[i])
        addi    a5,a5,4     # a5 = &A[i+1] ∈ increase address
        bne     a5,a3,.L3   # a5 != a3(=&A[N])? goto L3
        ret                 # return
.L4:
        li      a0,0        # sum = 0
        ret                 # return
```

N=0 case.




# Module Summary

# Module Summary

- **No high-level control flow constructs exist in assembly**
  - no for, no while, no do-while loops
  - not even if-then-else
- **Instead, all control flow implemented with goto instructions**
  - aka branch / jump instructions
  - conditional branch instructions only execute if the condition holds
  - similar to “if (cond) then goto X”
- **Translation of loop constructs**
  - for → while → do-while → if-then goto → assembly

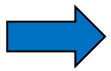
# Module Summary: RISC-V Branch Operations

| Conditional branch | Implemented as | Branch if | Signedness |
|---|---|---|---|
| `beq rs, rt, <lbl>` | `native` | == | don't care |
| `bne rs, rt, <lbl>` | `native` | != | don't care |
| `bge[u] rs, rt, <lbl>` | `native` | >= | signed / unsigned |
| `blt[u] rs, rt, <lbl>` | `native` | < | signed / unsigned |
| `bgt[u] rs, rt, <lbl>` | `blt[u] rt, rs, <lbl>` | > | signed / unsigned |
| `ble[u] rs, rt, <lbl>` | `bge[u] rt, rs, <lbl>` | <= | signed / unsigned |
| `beqz rs, <lbl>` | `beq rs, x0, <lbl>` | rs == 0 | don't care |
| `bnez rs, <lbl>` | `bne rs, x0, <lbl>` | rs != 0 | don't care |
| `blez rs, <lbl>` | `bge x0, rs, <lbl>` | rs <= 0 | signed |
| `bgez rs, <lbl>` | `bge rs, x0, <lbl>` | rs >= 0 | signed |
| `bltz rs, <lbl>` | `blt rs, x0, <lbl>` | rs < 0 | signed |
| `bgtz rs, <lbl>` | `blt x0, rs, <lbl>` | rs > 0 | signed |

# Module Summary: Loop Constructs

- **Do-While**

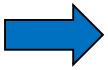
```
do
  body;
while (test);
```



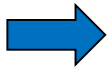
```
loop:
  body;
  if (test) goto loop;
```

- **While**

```
while (test)
  body;
```



```
  if (!test) goto done;
  do
    body;
  while (test);
done:
```

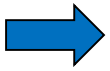


```
  if (!test) goto done;
loop:
  body;
  if (test) goto loop;
done:
```

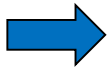
- **For**

```
for(init; test; update)
  body;
```

```
init;
while (test) {
  body;
  update;
}
```



```
  init;
  if (!test) goto done;
  do {
    body;
    update;
  } while (test);
done:
```



```
  init;
  if (!test) goto done;
loop:
  body;
  update;
  if (test) goto loop;
done:
```