Today

- Topic
    - Improve CPI by manually rearranging code
- Learning outcomes
    - Rearrange y86 instructions to improve CPI (i.e., practice for Lab 5)

Let’s try getting rid of bubbles

Assume forwarding and branch prediction (always taken)

```
    irmovq Data, %rbp                 # rbp points to data
    irmovq 8, %rsi                    # rsi = increment ptr
    xorq %rcx, %rcx                   # rcx = 0 (accumulator)
loop:
    mrmovq 0(%rbp), %rdi              # read from array
    andq %rdi, %rdi                   # Check for 0 element
    je Done                           # Done if 0
    addq %rdi, %rcx                   # add element to rcx
    addq %rsi, %rbp                   # advance to next item
    jmp loop                          # process next
Done:
    halt
Data:
    .quad 0xDECADE
    .quad 0xBE11
    .quad 0xAB1E
    .quad 0xABBA
    .quad 0
```

```
long long array[5];
long long *ap = array;
long long sum = 0;

while (*ap != 0) {
            sum += *ap;
            ap++;
}
```

1. How many bubbles?

2. CPI (Hint: 31 instructions are executed in the sequential case)

Hazards: Data

Assume forwarding and branch prediction (always taken)

```
    irmovq Data, %rbp              # rbp points to data
    irmovq 8, %rsi                 # rsi = 8
    xorq %rcx, %rcx                # rcx = 0 (accumulator)
loop:
    mrmovq 0(%rbp), %rdi           # read from array
    andq %rdi, %rdi                # Check for 0 element
    je Done                        # Done if 0
    addq %rdi, %rcx                # add element to rcx
    addq %rsi, %rbp                # advance to next item
    jmp loop                       # process next
Done:
    halt
Data:
    .quad 0xDECADE
    .quad 0xBE11
    .quad 0xAB1E
    .quad 0xABBA
    .quad 0
```

```
long long array[5];
long long *ap = array;
long long sum = 0;

while (*ap != 0) {
            sum += *ap;
            ap++;
}
```

1. How many bubbles?

   1 stall between memory and ALU op (5 iterations) = 5

2. CPI (Hint: 31 instructions are executed in the sequential case)

Hazards: Control

```
    irmovq Data, %rbp           # rbp points to data
    irmovq 8, %rsi              # rsi = 8
    xorq %rcx, %rcx             # rcx = 0 (accumulator)
loop:
    mrmovq 0(%rbp), %rdi        # read from array
    andq %rdi, %rdi             # Check for 0 element
    je Done                     # Done if 0
    addq %rdi, %rcx             # add element to rcx
    addq %rsi, %rbp             # advance to next item
    jmp loop                    # process next
Done:
    halt
Data:
    .quad 0xDECADE
    .quad 0xBE11
    .quad 0xAB1E
    .quad 0xABBA
    .quad 0
```

Assume forwarding and branch prediction (always taken)

```
long long array[5];
long long *ap = array;
long long sum = 0;

while (*ap != 0) {
        sum += *ap;
        ap++;
}
```

1. How many bubbles?
   1 stall between memory and ALU op (5 iterations) = 5
   2 quashed instructions for mispredicts (4 interations) = 8
2. CPI (Hint: 31 instructions are executed in the sequential case)

Hazards: CPI

Assume forwarding and branch prediction (always taken)

```
    irmovq Data, %rbp           # rbp points to data
    irmovq 8, %rsi              # rsi = 8
    xorq %rcx, %rcx             # rcx = 0 (accumulator)
loop:
    mrmovq 0(%rbp), %rdi        # read from array
    andq %rdi, %rdi             # Check for 0 element
    je Done                     # Done if 0
    addq %rdi, %rcx             # add element to rcx
    addq %rsi, %rbp             # advance to next item
    jmp loop                    # process next
Done:
    halt
Data:
    .quad 0xDECADE
    .quad 0xBE11
    .quad 0xAB1E
    .quad 0xABBA
    .quad 0
```

```
long long array[5];
long long *ap = array;
long long sum = 0;

while (*ap != 0) {
            sum += *ap;
            ap++;
}
```

1. How many bubbles?
   1 stall between memory and ALU op (5 iterations) = 5
   2 quashed instructions for mispredicts (4 interations) = 8
2. CPI (Hint: 31 instructions are executed in the sequential case)
   31 instructions + 13 stall/squash + 4 to fill pipe = 48
   CPI = 1.5

Can we do better?

Assume forwarding and branch prediction (always taken)

```
    irmovq Data, %rbp               # rbp points to data
    irmovq 8, %rsi                  # rsi = 8
    xorq %rcx, %rcx                 # rcx = 0 (accumulator)
loop:
    mrmovq 0(%rbp), %rdi            # read from array
    andq %rdi, %rdi                 # Check for 0 element
    je Done                         # Done if 0
    addq %rdi, %rcx                 # add element to rcx
    addq %rsi, %rbp                 # advance to next item
    jmp loop                        # process next
Done:
    halt
Data:
    .quad 0xDECADE
    .quad 0xBE11
    .quad 0xAB1E
    .quad 0xABBA
    .quad 0
```

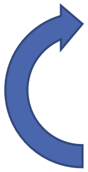
```
long long array[5];
long long *ap = array;
long long sum = 0;

while (*ap != 0) {
            sum += *ap;
            ap++;
}
```

There are two ways you can reduce the number of bubbles. What are they?

Let’s try getting rid of bubbles

Assume forwarding and branch prediction (always taken)



```
    irmovq Data, %rbp              # rbp points to data
    irmovq 8, %rsi                 # rsi = 8
    xorq %rcx, %rcx                # rcx = 0 (accumulator)
loop:
    mrmovq 0(%rbp), %rdi           # read from array
    andq %rdi, %rdi                # Check for 0 element
    je Done                        # Done if 0
    addq %rdi, %rcx                # add element to rcx
    addq %rsi, %rbp                # advance to next item
    jmp loop                       # process next
Done:
    halt
Data:
    .quad 0xDECADE
    .quad 0xBE11
    .quad 0xAB1E
    .quad 0xABBA
    .quad 0
```

```
long long array[5];
long long *ap = array;
long long sum = 0;

while (*ap != 0) {
            sum += *ap;
            ap++;
}
```

There are two ways you can reduce the number of bubbles. What are they?

1. Do something useful after the mrmovq that does not depend on rdi

Changing our jump

Assume forwarding and branch prediction (always taken)

```
    irmovq Data, %rbp               # rbp points to data
    irmovq 8, %rsi                  # rsi = 8
    xorq %rcx, %rcx                 # rcx = 0 (accumulator)
loop:
    mrmovq 0(%rbp), %rdi            # read from array
    addq %rsi, %rbp                 # advance to next item
    andq %rdi, %rdi                 # Check for 0 element
    je Done                         # exit
    addq %rdi, %rcx                 # add element to rcx
    jmp loop                        # process next
Done:
    halt
Data:
    .quad 0xDECADE
    .quad 0xBE11
    .quad 0xAB1E
    .quad 0xABBA
    .quad 0
```

```
long long array[5];
long long *ap = array;
long long sum = 0;

while (*ap != 0) {
            sum += *ap;
            ap++;
}
```

Can we put the conditional jump at the end of the loop and branch back?

There are two ways you can reduce the number of bubbles. What are they?

1. Do something useful after the mrmovq that does not depend on rdi
2. Can we turn this into a looping branch instead of two branches?

Changing our jump

Assume forwarding and branch prediction (always taken)

```
    irmovq Data, %rbp               # rbp points to data
    irmovq 8, %rsi                  # rsi = 8
    xorq %rcx, %rcx                 # rcx = 0 (accumulator)
loop:
    mrmovq 0(%rbp), %rdi            # read from array
    addq %rsi, %rbp                 # advance to next item
    andq %rdi, %rdi                 # Check for 0 element
~~    je Done                         # exit~~
    addq %rdi, %rcx                 # add element to rcx
    jmp loop                        # process next
Done:
    halt
Data:
    .quad 0xDECADE
    .quad 0xBE11
    .quad 0xAB1E
    .quad 0xABBA
    .quad 0
```

```
long long array[5];
long long *ap = array;
long long sum = 0;

while (*ap != 0) {
            sum += *ap;
            ap++;
}
```

We want this to be a JNE

There are two ways you can reduce the number of bubbles. What are they?

1. Do something useful after the mrmovq that does not depend on rdi
2. Can we turn this into a looping branch instead of two branches?

How many bubbles are left?

Assume forwarding and branch prediction (always taken)

```
    irmovq Data, %rbp            # rbp points to data
    irmovq 8, %rsi               # rsi = 8
    xorq %rcx, %rcx              # rcx = 0 (accumulator)
loop:
    mrmovq 0(%rbp), %rdi         # read from array
    addq %rsi, %rbp              # advance to next item
    addq %rdi, %rcx              # add element to rcx
    andq %rdi, %rdi              # Check for 0 element
    jne loop                     # process next
Done:
    halt
Data:
    .quad 0xDECADE
    .quad 0xBE11
    .quad 0xAB1E
    .quad 0xABBA
    .quad 0
```

```
long long array[5];
long long *ap = array;
long long sum = 0;

while (*ap != 0) {
          sum += *ap;
          ap++;
}
```

There are two ways you can reduce the number of bubbles. What are they?

1. Do something useful after the mrmovq that does not depend on rdi
2. Can we turn this into a looping branch instead of two branches?

29 instructions in 35 cycles => 1.2 CPI!