

DOWNLOAD & USE **THE GOTO GUIDE APP**

CREATE YOUR SCHEDULE
NETWORK WITH OTHERS
ASK QUESTIONS
RATE SESSIONS



What Every Coder Should Know About How CPUs Work

Matt Godbolt @mattgodbolt

21st October 2024

What should you get out of this?

- Overview of modern CPUs
- How compilers take advantage
- Tooling for when it goes wrong

A modern CPU pipeline

(simplified)

- In-order front-end
- Out-of-order back-end
- In-order retirement

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Branch Prediction

- Intelligent guess
- Is there a branch coming up?
- Where does it go?
- Will it be taken?

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Front end

- Reads instruction bytes
- Decodes them into internal μ ops
- Rewrites them to avoid data hazards

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Back-end

- Load/Store Units
- Address Generators
- ALUs
- FPU's
- Vector units

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Retirement

- "Commits"
- In program order
- Exceptions handled

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Branch Prediction

Inputs

Instruction address

7ffe3345610

Branch History

0111101101100

BTB

Tag	Dest Addr
7ffe334	7ffe33455fc
7ffe11d	7fff917350b
...	

Predictor(s)

   
   
...
   

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Branch Prediction

```
1 def process(list_of_ints: list[int]) -> (int, int):
2     sum_of_elements_below_128 = 0
3     total_sum = 0
4
5     for integer in list_of_ints:
6         if integer < 128:
7             sum_of_elements_below_128 += integer
8             total_sum += integer
9
10    return sum_of_elements_below_128, total_sum
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Branch Prediction

```
1 def process(list_of_ints: list[int]) -> (int, int):
2     sum_of_elements_below_128 = 0
3     total_sum = 0
4
5     for integer in list_of_ints:
6         if integer < 128:
7             sum_of_elements_below_128 += integer
8             total_sum += integer
9
10    return sum_of_elements_below_128, total_sum
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Branch Prediction

```
1 def process(list_of_ints: list[int]) -> (int, int):
2     sum_of_elements_below_128 = 0
3     total_sum = 0
4
5     for integer in list_of_ints:
6         if integer < 128:
7             sum_of_elements_below_128 += integer
8             total_sum += integer
9
10    return sum_of_elements_below_128, total_sum
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Branch Prediction

```
1 def process(list_of_ints: list[int]) -> (int, int):
2     sum_of_elements_below_128 = 0
3     total_sum = 0
4
5     for integer in list_of_ints:
6         if integer < 128:
7             sum_of_elements_below_128 += integer
8             total_sum += integer
9
10    return sum_of_elements_below_128, total_sum
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Branch Prediction

```
1 def process(list_of_ints: list[int]) -> (int, int):
2     sum_of_elements_below_128 = 0
3     total_sum = 0
4
5     for integer in list_of_ints:
6         if integer < 128:
7             sum_of_elements_below_128 += integer
8             total_sum += integer
9
10    return sum_of_elements_below_128, total_sum
```

- Run with random data
- Run with same data, but sorted

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Results for ten million items

- Unsorted: $0.70s \pm 0.02s$

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Results for ten million items

- Unsorted: $0.70s \pm 0.02s$
- Sorted: $0.62s \pm 0.01s$

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Results for ten million items

- Unsorted: $0.70s \pm 0.02s$
- Sorted: $0.62s \pm 0.01s$

Is it the branch predictor?

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Enter perf

```
$ perf stat -e branches,branch-misses -- python3 BPU.py --unsorted  
...  
20,074,920,709 branches  
124,836,299 branch-misses # 0.62% of all branches
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Enter perf

```
$ perf stat -e branches,branch-misses -- python3 BPU.py --unsorted
```

```
...
```

```
20,074,920,709 branches
```

```
124,836,299 branch-misses # 0.62% of all branches
```

```
$ perf stat -e branches,branch-misses -- python3 BPU.py --sorted
```

```
...
```

```
19,962,912,393 branches
```

```
73,880,827 branch-misses # 0.37% of all branches
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

What about C++??

```
1 for (auto integer : list_of_ints) {  
2     total_sum += integer;  
3     if (integer < 128) sum_below_128 += integer;  
4 }
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Results for ten million items (C++)

- Unsorted: 7.6ms $\pm$ 0.4ms

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Results for ten million items (C++)

- Unsorted: 7.6ms $\pm$ 0.4ms
- Sorted: 7.6ms $\pm$ 0.4ms

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

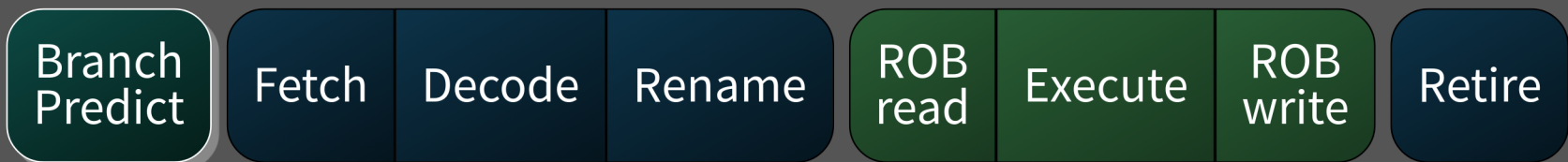
ROB
write

Retire

Results for ten million items (C++)

- Unsorted: 7.6ms $\pm$ 0.4ms
- Sorted: 7.6ms $\pm$ 0.4ms

What happened?



Checking C++ in perf

```
$ perf stat -e branches,branch-misses -- ./bpu --unsorted  
...  
5,471,740,896    branches  
38,953,913      branch-misses    #    0.71% of all branches
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Checking C++ in perf

```
$ perf stat -e branches,branch-misses -- ./bpu --unsorted
```

```
...  
5,471,740,896    branches  
38,953,913      branch-misses    #    0.71% of all branches
```

```
$ perf stat -e branches,branch-misses -- ./bpu --sorted
```

```
...  
5,471,064,238    branches  
38,891,885      branch-misses    #    0.71% of all branches
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

What's going on?

```
for (auto integer : list_of_ints) {  
    total_sum += integer;  
    if (integer < 128) sum_below_128 += integer;  
}
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

What's going on?

```
for (auto integer : list_of_ints) {  
    total_sum += integer;  
    if (integer < 128) sum_below_128 += integer;  
}
```

```
1 .L4:  
2  mov    ecx, [rdx]      ; ecx is 'integer'  
3  add    r8d, ecx        ; total_sum += integer  
4  lea    r9d, [rsi+rcx]   ; temp = sum_below_128 + integer  
5  cmp    ecx, 128        ; compare with 128  
6  cmovb  esi, r9d        ; sum_below_128 = temp if integer < 128  
7  add    rdx, 4          ; move to next integer  
8  cmp    r10, rdx        ; compare with end of list  
9  jne    .L4            ; loop if not done
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

What's going on?

```
for (auto integer : list_of_ints) {  
    total_sum += integer;  
    if (integer < 128) sum_below_128 += integer;  
}
```

```
1 .L4:  
2  mov    ecx, [rdx]      ; ecx is 'integer'  
3  add    r8d, ecx        ; total_sum += integer  
4  lea    r9d, [rsi+rcx]   ; temp = sum_below_128 + integer  
5  cmp    ecx, 128        ; compare with 128  
6  cmovb  esi, r9d        ; sum_below_128 = temp if integer < 128  
7  add    rdx, 4          ; move to next integer  
8  cmp    r10, rdx        ; compare with end of list  
9  jne    .L4            ; loop if not done
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

What's going on?

```
for (auto integer : list_of_ints) {  
    total_sum += integer;  
    if (integer < 128) sum_below_128 += integer;  
}
```

```
1 .L4:  
2  mov    ecx, [rdx]      ; ecx is 'integer'  
3  add    r8d, ecx        ; total_sum += integer  
4  lea    r9d, [rsi+rcx]   ; temp = sum_below_128 + integer  
5  cmp    ecx, 128        ; compare with 128  
6  cmovb  esi, r9d        ; sum_below_128 = temp if integer < 128  
7  add    rdx, 4          ; move to next integer  
8  cmp    r10, rdx        ; compare with end of list  
9  jne    .L4            ; loop if not done
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

What's going on?

```
for (auto integer : list_of_ints) {  
    total_sum += integer;  
    if (integer < 128) sum_below_128 += integer;  
}
```

```
1 .L4:  
2  mov    ecx, [rdx]      ; ecx is 'integer'  
3  add    r8d, ecx        ; total_sum += integer  
4  lea    r9d, [rsi+rcx]   ; temp = sum_below_128 + integer  
5  cmp    ecx, 128        ; compare with 128  
6  cmovb  esi, r9d        ; sum_below_128 = temp if integer < 128  
7  add    rdx, 4          ; move to next integer  
8  cmp    r10, rdx        ; compare with end of list  
9  jne    .L4            ; loop if not done
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

What's going on?

```
for (auto integer : list_of_ints) {  
    total_sum += integer;  
    if (integer < 128) sum_below_128 += integer;  
}
```

```
1 .L4:  
2  mov    ecx, [rdx]      ; ecx is 'integer'  
3  add    r8d, ecx        ; total_sum += integer  
4  lea    r9d, [rsi+rcx]   ; temp = sum_below_128 + integer  
5  cmp    ecx, 128        ; compare with 128  
6  cmovb  esi, r9d        ; sum_below_128 = temp if integer < 128  
7  add    rdx, 4          ; move to next integer  
8  cmp    r10, rdx        ; compare with end of list  
9  jne    .L4            ; loop if not done
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

What's going on?

```
for (auto integer : list_of_ints) {  
    total_sum += integer;  
    if (integer < 128) sum_below_128 += integer;  
}
```

```
1 .L4:  
2  mov    ecx, [rdx]      ; ecx is 'integer'  
3  add    r8d, ecx        ; total_sum += integer  
4  lea    r9d, [rsi+rcx]   ; temp = sum_below_128 + integer  
5  cmp    ecx, 128        ; compare with 128  
6  cmovb  esi, r9d        ; sum_below_128 = temp if integer < 128  
7  add    rdx, 4          ; move to next integer  
8  cmp    r10, rdx        ; compare with end of list  
9  jne    .L4            ; loop if not done
```

The compiler is clever

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

If the compiler was dumb

```
// Compiled with -fno-if-conversion to disable cmov
for (auto integer : list_of_ints) {
    total_sum += integer;
    if (integer < 128) sum_below_128 += integer;
}
```

```
$ perf stat -e branches,branch-misses -- ./bpu-no-cmov --sorted
10,472,350,723    branches
   38,942,794    branch-misses    #    0.37% of all branches
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

If the compiler was dumb

```
// Compiled with -fno-if-conversion to disable cmov
for (auto integer : list_of_ints) {
    total_sum += integer;
    if (integer < 128) sum_below_128 += integer;
}
```

```
$ perf stat -e branches,branch-misses -- ./bpu-no-cmov --sorted
10,472,350,723    branches
   38,942,794    branch-misses    #    0.37% of all branches
```

```
$ perf stat -e branches,branch-misses -- ./bpu-no-cmov --unsorted
10,495,074,964    branches
  2,540,309,916    branch-misses    #   24.20% of all branches
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

If the compiler was dumb

```
// Compiled with -fno-if-conversion to disable cmov
for (auto integer : list_of_ints) {
    total_sum += integer;
    if (integer < 128) sum_below_128 += integer;
}
```

```
$ perf stat -e branches,branch-misses -- ./bpu-no-cmov --sorted
10,472,350,723    branches
   38,942,794    branch-misses    #    0.37% of all branches
```

```
$ perf stat -e branches,branch-misses -- ./bpu-no-cmov --unsorted
10,495,074,964    branches
  2,540,309,916    branch-misses    #   24.20% of all branches
```

39.7ms $\pm$ 2.3ms (vs 9.3ms / 7.6ms)

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Execution



Branch
Predict

Fetch

Decode

Rename

ROB
read

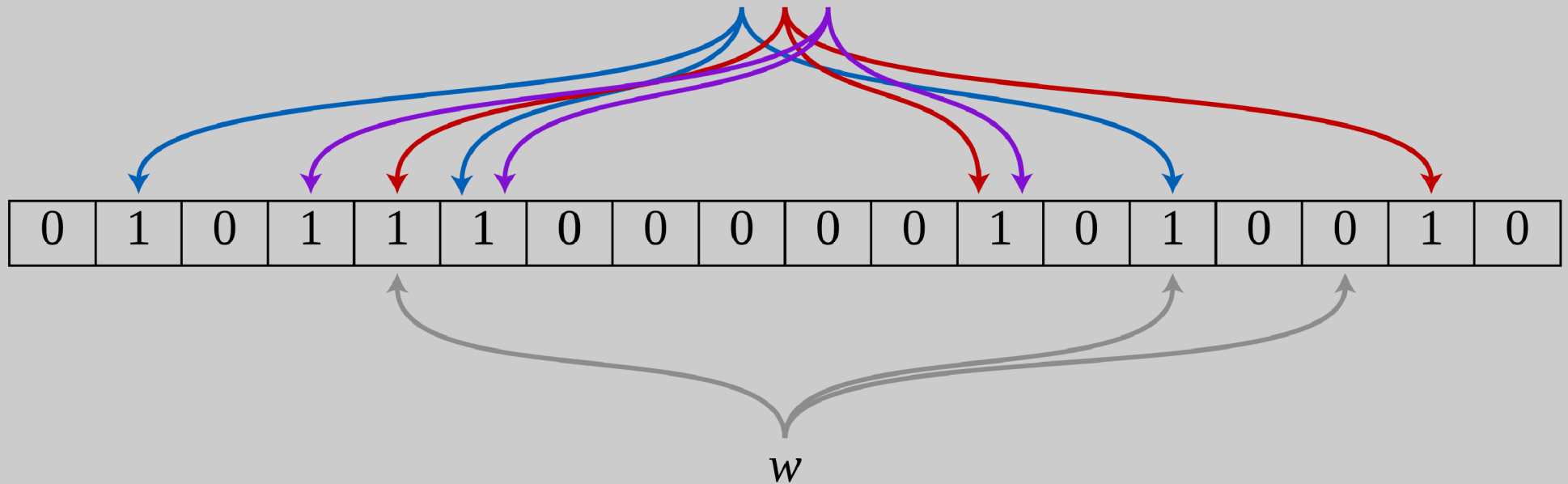
Execute

ROB
write

Retire

Bloom Filter

$\{x, y, z\}$



Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Bloom Filter

```
1 struct BloomFilter {
2     size_t bucket_for(size_t hash) const {
3         return hash % filter_.size();
4     }
5     bool maybe_present(unsigned id) const {
6         size_t count = 0;
7         for (size_t i = 0; i < NumHashes; ++i) {
8             const auto hash = scramble(id ^ i);
9             count += filter_[bucket_for(hash)];
10        }
11        return count == NumHashes;
12    }
13    std::vector<char> filter_;
14 };
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Bloom Filter

```
1 struct BloomFilter {
2     size_t bucket_for(size_t hash) const {
3         return hash % filter_.size();
4     }
5     bool maybe_present(unsigned id) const {
6         size_t count = 0;
7         for (size_t i = 0; i < NumHashes; ++i) {
8             const auto hash = scramble(id ^ i);
9             count += filter_[bucket_for(hash)];
10        }
11        return count == NumHashes;
12    }
13    std::vector<char> filter_;
14 };
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Bloom Filter

```
1 struct BloomFilter {
2     size_t bucket_for(size_t hash) const {
3         return hash % filter_.size();
4     }
5     bool maybe_present(unsigned id) const {
6         size_t count = 0;
7         for (size_t i = 0; i < NumHashes; ++i) {
8             const auto hash = scramble(id ^ i);
9             count += filter_[bucket_for(hash)];
10        }
11        return count == NumHashes;
12    }
13    std::vector<char> filter_;
14 };
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Bloom Filter

```
1 struct BloomFilter {
2     size_t bucket_for(size_t hash) const {
3         return hash % filter_.size();
4     }
5     bool maybe_present(unsigned id) const {
6         size_t count = 0;
7         for (size_t i = 0; i < NumHashes; ++i) {
8             const auto hash = scramble(id ^ i);
9             count += filter_[bucket_for(hash)];
10        }
11        return count == NumHashes;
12    }
13    std::vector<char> filter_;
14 };
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Bloom Filter

```
1 struct BloomFilter {  
2     size_t bucket_for(size_t hash) const {  
3         return hash % filter_.size();  
4     }  
5     bool maybe_present(unsigned id) const {  
6         size_t count = 0;  
7         for (size_t i = 0; i < NumHashes; ++i) {  
8             const auto hash = scramble(id ^ i);  
9             count += filter_[bucket_for(hash)];  
10        }  
11        return count == NumHashes;  
12    }  
13    std::vector<char> filter_;  
14 };
```

[Analysis Link](#)

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Execution

```
1 struct BloomFilter {  
2     explicit BloomFilter(size_t pow2_size) : filter_(1 << pow2_size) {  
3     }  
4     size_t bucket_for(size_t hash) const {  
5         return hash & (filter_.size() - 1);  
6     }  
7     bool maybe_present(unsigned id) const {  
8         // as before...
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Execution

```
1 struct BloomFilter {  
2     explicit BloomFilter(size_t pow2_size) : filter_(1 << pow2_size) {  
3     }  
4     size_t bucket_for(size_t hash) const {  
5         return hash & (filter_.size() - 1);  
6     }  
7     bool maybe_present(unsigned id) const {  
8         // as before...
```

Analysis Link

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Execution

```
1 struct BloomFilter {  
2     explicit BloomFilter(size_t pow2_size) : filter_(1 << pow2_size) {  
3     }  
4     size_t bucket_for(size_t hash) const {  
5         return hash & (filter_.size() - 1);  
6     }  
7     bool maybe_present(unsigned id) const {  
8         // as before...
```

[Analysis Link](#)

30 cycles (vs 148)

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Can the compiler save us?

```
1 size_t bucket_for(size_t hash) {  
2     return hash % 1021u;  
3 }
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Can the compiler save us?

```
1 size_t bucket_for(size_t hash) {  
2     return hash % 1021u;  
3 }
```

libdivide

Branch
Predict

Fetch

Decode

Rename

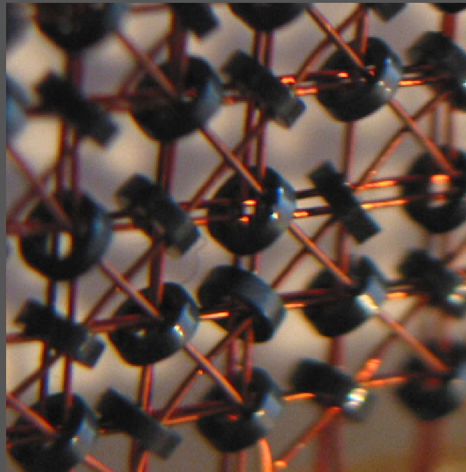
ROB
read

Execute

ROB
write

Retire

Memory



Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Memory

Name	Size (KiB)	Access time (ns)
L1 (data)	32	1
L2	256	5
L3	8000	15-20
RAM	>GBs	80-120+

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Memory

```
1 struct Element {  
2     Element *next;  
3     size_t data;  
4 };  
5  
6 size_t sum_linked(const Element *first) {  
7     size_t total = 0;  
8     for (auto elem = first; elem; elem = elem->next) {  
9         total += elem->data;  
10    }  
11    return total;  
12 }
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Memory

```
1 struct Element {  
2     Element *next;  
3     size_t data;  
4 };  
5  
6 size_t sum_linked(const Element *first) {  
7     size_t total = 0;  
8     for (auto elem = first; elem; elem = elem->next) {  
9         total += elem->data;  
10    }  
11    return total;  
12 }
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Memory

```
1 struct Element {  
2     Element *next;  
3     size_t data;  
4 };  
5  
6 size_t sum_linked(const Element *first) {  
7     size_t total = 0;  
8     for (auto elem = first; elem; elem = elem->next) {  
9         total += elem->data;  
10    }  
11    return total;  
12 }
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Memory

```
1 struct Element {  
2     Element *next;  
3     size_t data;  
4 };  
5  
6 size_t sum_linked(const Element *first) {  
7     size_t total = 0;  
8     for (auto elem = first; elem; elem = elem->next) {  
9         total += elem->data;  
10    }  
11    return total;  
12 }
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Results

10M elements / 150MBytes

- Array: 11.7ms $\pm$ 1.4ms

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Results

10M elements / 150MBytes

- Array: 11.7ms $\pm$ 1.4ms
- Linked list: 24.6ms $\pm$ 2.7ms

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Results

10M elements / 150MBytes

- Array: 11.7ms $\pm$ 1.4ms
- Linked list: 24.6ms $\pm$ 2.7ms
- Only 2.1x slower

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Results

10M elements / 150MBytes

- Array: 11.7ms $\pm$ 1.4ms
- Linked list: 24.6ms $\pm$ 2.7ms
- Only 2.1x slower

...but I cheated

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

"Real"? Results

Randomized link order

Linear array	11.7ms $\pm$ 1.4ms
Linked list (linear)	24.6ms $\pm$ 2.7ms
Linked list (random)	1,301ms $\pm$ 29.3ms

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Top-Down Analysis

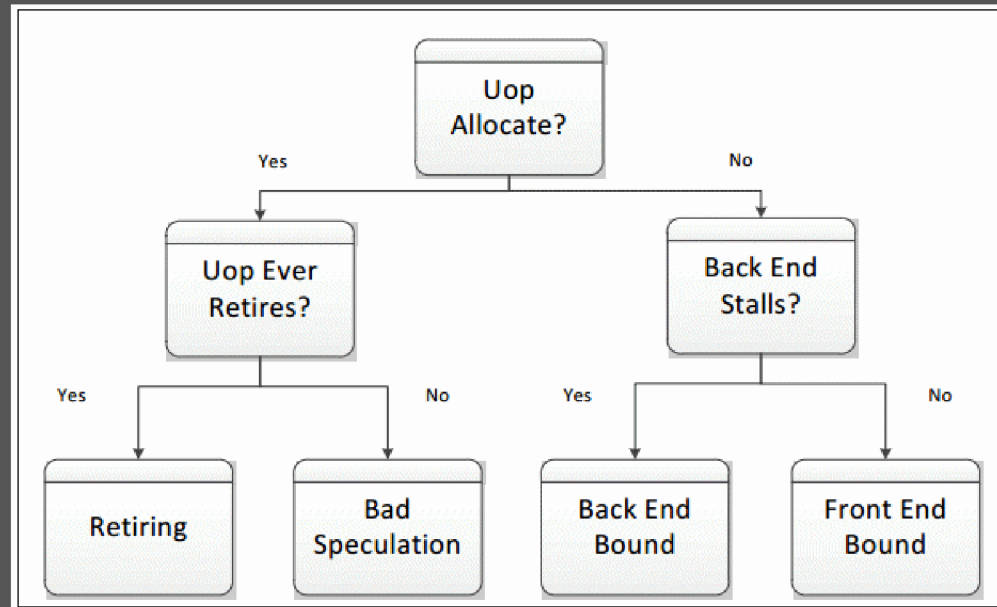


Image from Intel's web site

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Perf

```
$ perf stat --topdown -- ./mem 1000000 --array  
%frontend_bound %retiring %backend_bound %bad_speculation  
4.3 27.2 68.8 0.3
```

```
$ perf stat --topdown -- ./mem 1000000 --linked  
%frontend_bound %retiring %backend_bound %bad_speculation  
2.3 11.8 85.1 0.8
```

```
$ perf stat --topdown -- ./mem 1000000 --linked --random  
%frontend_bound %retiring %backend_bound %bad_speculation  
0.5 1.2 98.3 0.1
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Perf - drilling down

```
$ perf stat --topdown --td-level=2 -- ./mem 1000000 --linked --random
%heavy_ops %light_ops %fetch_lat %fetch_bw %mispredicts %memory_bound
0.1 1.0 0.5 0.2 0.1 95.6
```

```
$ perf stat --topdown -M tma_memory_bound_group ...
%dram_bound %store_bound %l1_bound %l2_bound %l3_bound
67.4 0.0 12.3 0.6 15.2
```

Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire

Conclusion

Conclusion

- CPUs are complex
 - Pipelines
 - Out of Order
 - Branch prediction
 - Memory & Caches
- Compilers are smart
- Use tools to understand

What we didn't cover

- SMT (HyperThread)
- Multiple cores
- Cache coherency
- SIMD
- kcachegrind
- uICA

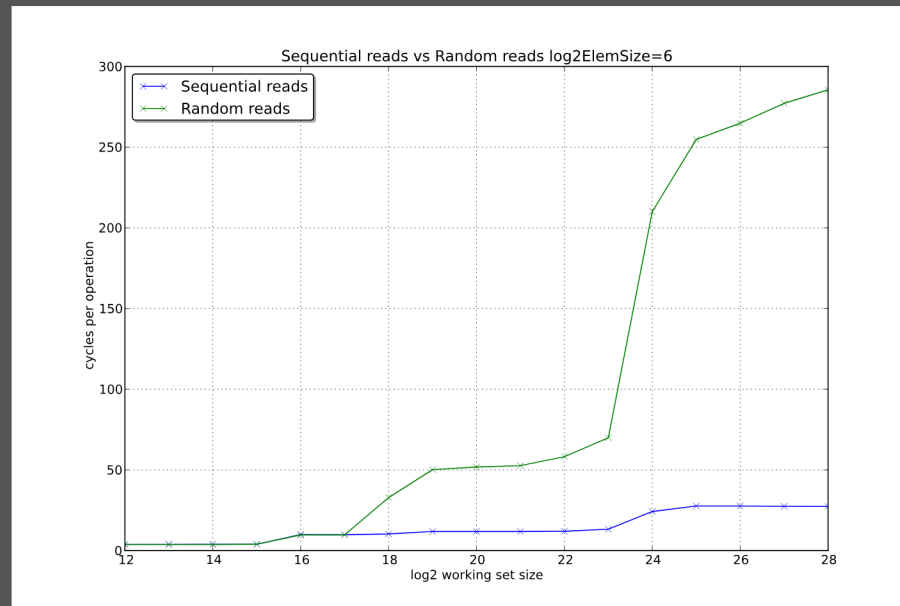
Questions?

Matt Godbolt
[@mattgodbolt](https://twitter.com/mattgodbolt)

Rate this session in the GOTO Guide app

Help us make GOTO even better

Real-world measurements



Branch
Predict

Fetch

Decode

Rename

ROB
read

Execute

ROB
write

Retire