What we’re adding (quick view)

1. A command-line option to select a priority policy.
2. A priority field on each RUU entry.
3. Logic to compute priority when an op becomes ready.
4. A small change to ready queue ordering so higher priority issues first.
5. A couple of stats + prints so you can see the effect in out_priority.txt.

Here run like this:

./sim-outorder -issue:policy priority -redir:sim out_priority.txt tests/bin.little/test-math

If we omit -issue:policy, it will behave like before.

1) Add an option: -issue:policy

Where: sim_reg_options(...) — around line ~574

Search for: sim_reg_options and you’ll see lots of opt_reg_* calls. Added this block with the others:

```
/* issue selection policy: "default" or "priority" */
static char *issue_policy = "default";
```

Register it:

```
opt_reg_string(odb, "-issue:policy",
```

```
"issue selection policy {default|priority}",
&issue_policy, /* default */ "default",
/* print */ TRUE, /* format */ NULL);
```

right after the -issue:width option inside

sim_reg_options(...). That's it for the flag.

2) Add a priority field to each instruction (RUU entry)

Where: struct RUU_station — around line ~1490

Search for: struct RUU_station {. Inside that struct, added a field near the other small ints: int priority;

```
/* computed issue priority: higher value = higher priority */
```

This won't change behavior yet; it just stores the number we'll compute later.

3) Decide how to compute priority

We'll use a simple, effective rule:

- Give loads, long-latency ops, and branches a bump (they often gate progress).
- Prefer older ready instructions (smaller seq) when priorities tie (keeps forward progress).

Simple priority rules we use

Instruction gets points as follows:

Instruction type Priority bonus

Long-latency ops (mult/div) +4
Branches (control ops) +4
Loads/Stores (memory) +3

So example:

- Load → priority = 3
- Branch → 4
- Multiply → 4

Older instructions (smaller seq number) win ties.

Added a tiny helper above readyq_enqueue(...). For example, just before the readyq_enqueue function (a few lines above ~1988 is fine):

```
/* Compute a simple static/dynamic priority for ready instructions */

static inline int

compute_issue_priority(struct RUU_station *rs)

{
  int p = 0;
 /* Long latency ALU/mul/div or control flow gets a bump */

  if (MD_OP_FLAGS(rs->op) & (F_LONGLAT | F_CTRL)) p += 4;

 /* Memory ops (LSQ participants) also matter */

  if (rs->in_LSQ) p += 3;


  return p;
```

}

Now, at the start of readyq_enqueue(struct RUU_station *rs) (right after the existing “node is now queued” check), set:

```
/* establish priority if we are in priority policy */
  if (!strcmp(issue_policy, "priority"))
    rs->priority = compute_issue_priority(rs);
  else
    rs->priority = 0; /* not used in default policy */
```

4) Change the ready-queue ordering (only when policy==priority)

Where: still in readyq_enqueue(...) — around lines ~2006–2026

Here see the existing insertion logic:

```
  if (rs->in_LSQ || MD_OP_FLAGS(rs->op) & (F_LONGLAT|F_CTRL)) { /*
   insert loads/stores and long latency ops at the head of the queue */ prev =
   NULL;
    node = ready_queue;
  } else {
   /* otherwise insert in program order (earliest seq first) */
   for (prev=NULL, node=ready_queue;
        node && node->x.seq < rs->seq;
```

```
        prev=node, node=node->next);
  }
```

Wrap that with a policy check. Replaced that whole if (...) else {...} with this:

```
if (!strcmp(issue_policy, "priority")) {

    /* Insert by: higher priority first; tie-break by earlier seq */
    for (prev=NULL, node=ready_queue;
         node &&
           ( (RSLINK_RS(node)->priority > rs->priority) ||
             (RSLINK_RS(node)->priority == rs->priority && node->x.seq < rs->seq) );
         prev=node, node=node->next);

  } else {
    /* Original behavior */
     if (rs->in_LSQ || MD_OP_FLAGS(rs->op) & (F_LONGLAT|F_CTRL)) {
       prev = NULL;

       node = ready_queue;
    } else {
      for (prev=NULL, node=ready_queue;
           node && node->x.seq < rs->seq;
           prev=node, node=node->next);
    }
  }
```

This keeps default behavior unchanged but, under -issue:policy priority, the ready queue will be globally ordered by your computed priority, with a stable tie-break on age (seq).

Meaning:

- If priority mode, insert based on priority first, then age
- Else, keep original behavior

Visual Explanation

Imagine ready queue:

Current ready queue (priority mode):

[Priority 4] branch

[Priority 3] load

A new multiply comes (priority 4).

We insert it at the top, before branch.

[Priority 4] multiply → new highest

[Priority 4] branch

[Priority 3] load

CPU issues multiply first.

5) Track & print in out_priority.txt

A. Declared a counter near the other global counters
Where: top of file where other counter_t stats live — around lines ~870–930 (you'll see many static counter_t sim_*).

Added this:

```
/* how many instructions were issued from the highest-priority slot */
static counter_t sim_priority_issued = 0;
```

B. Bump it when we actually issue a highest-priority op

Where: ruu_issue(void) — function starts around line ~2611

as soon as you enter the main for (n_issued=0; node && n_issued < ruu_issue_width; ...) loop, detect if policy==priority and n_issued==0 and node is the top of the (already priority-sorted) list:

```
 for (n_issued=0; node && n_issued < ruu_issue_width; node = next_node) {

    next_node = node->next;


    if (!strcmp(issue_policy, "priority") && n_issued == 0) {

     /* We are looking at the head of the (already sorted) ready list */

     sim_priority_issued++;

   }
```

(Placed this just before the existing /* still valid? */ check is fine.)

C. Register and print the stat

Where: sim_reg_stats(struct stat_sdb_t *sdb) — around line ~1190

Added:

```
stat_reg_counter(sdb, "sim_priority_issued",

    "count of cycles that issued a head-of-queue (highest-priority) inst",
    &sim_priority_issued, 0, NULL);
```

Now when we run with -redir:sim out_priority.txt, this counter will show up in that file at the end of the

run. In order to get per-cycle traces:

Where: still in ruu_issue(void) inside the loop where we issue the first inst each cycle (the same spot as above).

Added:

```
if (!strcmp(issue_policy, "priority") && n_issued == 0) {
 /* Goes to -redir:sim output file */
 fprintf(stderr, "[cycle %llu] head prio=%d seq=%u op=%s\n",
         (unsigned long long)sim_cycle,
         RSLINK_RS(node)->priority,
         RSLINK_RS(node)->seq,
         MD_OP_NAME(RSLINK_RS(node)->op));
}
```

Because we run with -redir:sim out_priority.txt, those lines will land in out_priority.txt, letting us visually confirm the priority order each cycle.

6) Build and run

make clean && make sim-outorder

./sim-outorder -issue:policy priority -redir:sim out_priority.txt tests/bin.little/test-math

What to look for in out_priority.txt:

- The sim_priority_issued stat in the summary section.

- many lines like:

[cycle 1234] head prio=3 seq=456 op=LDQ

we can compare run with -issue:policy default vs priority to see differences in IPC and this new stat.

7) Final Output

With priority scheduler:

- Loads start earlier
- Branches handled faster
- Long ops issued sooner
- IPC may improve on some workloads

We will see priority decisions in out_priority.txt.

Very important reminder

To use this policy:

./sim-outorder -issue:policy priority -redir:sim out_priority.txt tests/bin/test-math
Without this flag, CPU behaves normally.

CHANGES

line 165

```
/* issue selection policy: "default" or "priority" */
static char *issue_policy = "default";
```

line 710

```
opt_reg_string(odb, "-issue:policy",
               "issue selection policy {default|priority}",

               &issue_policy, "default", NULL);
```

line 1506

```
  int priority; /* computed issue priority: higher value = higher priority */
```

line 1988

```
/* Compute a simple priority for ready instructions */

static inline int compute_issue_priority(struct RUU_station *rs)

{

  int p = 0;

 /* Long latency ALU/mul/div or control flow gets a bump */ if

  (MD_OP_FLAGS(rs->op) & (F_LONGLAT | F_CTRL)) p += 4;

  /* Memory ops (LSQ participants) also matter */

  if (rs->in_LSQ) p += 3;
```

```
 return p;
}


line 2027
 /* establish priority if we are in priority policy */

  if (!strcmp(issue_policy, "priority"))

    rs->priority = compute_issue_priority(rs);

  else

    rs->priority = 0; /* not used in default policy */


line 2043

  if (!strcmp(issue_policy, "priority")) {

   /* Insert by: higher priority first; tie-break by earlier seq */

   for (prev=NULL, node=ready_queue;

        node &&

          ( (RSLINK_RS(node)->priority > rs->priority) ||

            (RSLINK_RS(node)->priority == rs->priority && node->x.seq < rs->seq) );

        prev=node, node=node->next);

 } else {

   /* Original behavior */

    if (rs->in_LSQ || MD_OP_FLAGS(rs->op) & (F_LONGLAT|F_CTRL)) {

      prev = NULL;


      node = ready_queue;
```

```
    } else {
      for (prev=NULL, node=ready_queue;
           node && node->x.seq < rs->seq;
           prev=node, node=node->next);

    }
  }
```

line 309

```
/* how many instructions were issued from the highest-priority slot */

static counter_t sim_priority_issued = 0;
```

line 2686

```
  for (n_issued=0; node && n_issued < ruu_issue_width; node = next_node) {

     next_node = node->next;


     if (!strcmp(issue_policy, "priority") && n_issued == 0) {

      /* We are looking at the head of the (already sorted) ready list */

      sim_priority_issued++;


    }
```

line 1208

```
    stat_reg_counter(sdb, "sim_priority_issued",
    "count of cycles that issued a head-of-queue (highest-priority) inst",
```

```
    &sim_priority_issued, 0, NULL);


 line 2698
 if (!strcmp(issue_policy, "priority") && n_issued == 0) {
  /* Goes to -redir:sim output file */
  fprintf(stderr, "[cycle %llu] head prio=%d seq=%u op=%s\n",
          (unsigned long long)sim_cycle,
          RSLINK_RS(node)->priority,
          RSLINK_RS(node)->seq,
          MD_OP_NAME(RSLINK_RS(node)->op));
}
```