

Machine Scheduler

Fine grain resource allocation using **ResourceSegments**

Adam Nemet, Francesco Petrogalli (speaker), Francis Visoiu-Mistrih - Apple

What is this all about

- MachineScheduler and SchedMachineModel
- No ~~InstrItineraries~~
- Representation of hardware resources in the SchedMachineModel
- Improved estimates of execution traces
 - Better scheduling
- Ongoing effort

Background information

Definitions

Definitions

Instruction is
Ready @<cycle>

All input data needed by an
instruction is ready

Definitions

Instruction is
Ready @<cycle>

All input data needed by an
instruction is ready

```
ADD r2, r1, r0
ADD r5, r4, r3
MUL r6, r5, r2
```

Definitions

Instruction is
Ready @<cycle>

All input data needed by an
instruction is ready

```
ADD r2, r1, r0
ADD r5, r4, r3
MUL r6, r5, r2
```

Instruction is
Available @<cycle>

All hardware resources that execute
the instruction are available

Definitions

Instruction is
Ready @<cycle>

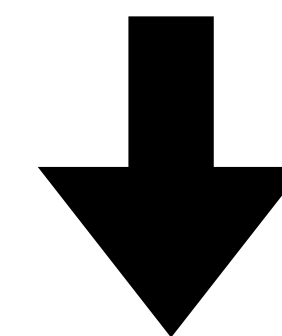
All input data needed by an
instruction is ready

```
ADD r2, r1, r0
ADD r5, r4, r3
MUL r6, r5, r2
```

Instruction is
Available @<cycle>

All hardware resources that execute
the instruction are available

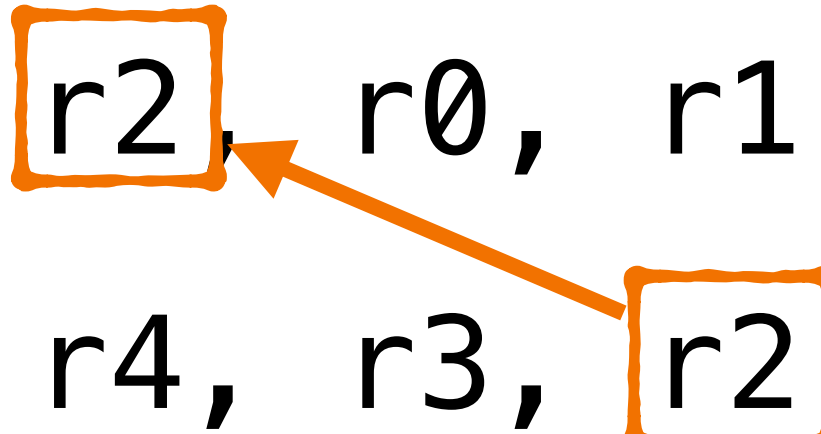
ADD r2, r1, r0



Adder

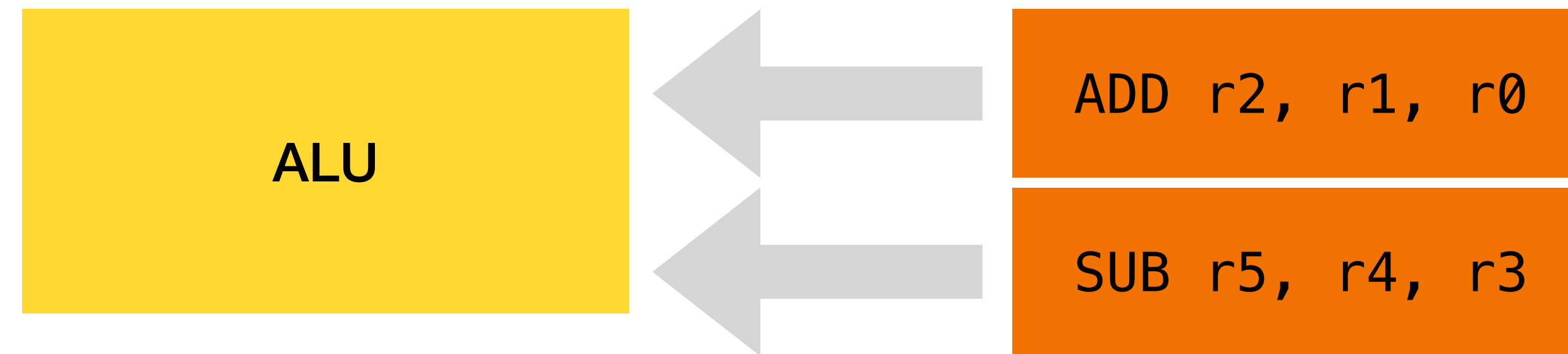
**All instructions used in this
presentation are **READY****

ADD r2, r0, r1
ADD r4, r3, r2



```
graph TD; A[ADD r2, r0, r1]; B[ADD r4, r3, r2]; B --> A;
```

Focus on structural hazards



Instructions breakdown

Instructions breakdown

Sequence of stages

Instructions breakdown

Sequence of stages



Instructions breakdown

Sequence of stages



Instructions breakdown

Sequence of stages



Instructions breakdown

Sequence of stages



Instructions used in this talk

ADD



MADD



Let's schedule some code!

The stage “Execute MADD” can process one instruction at a time

MADD



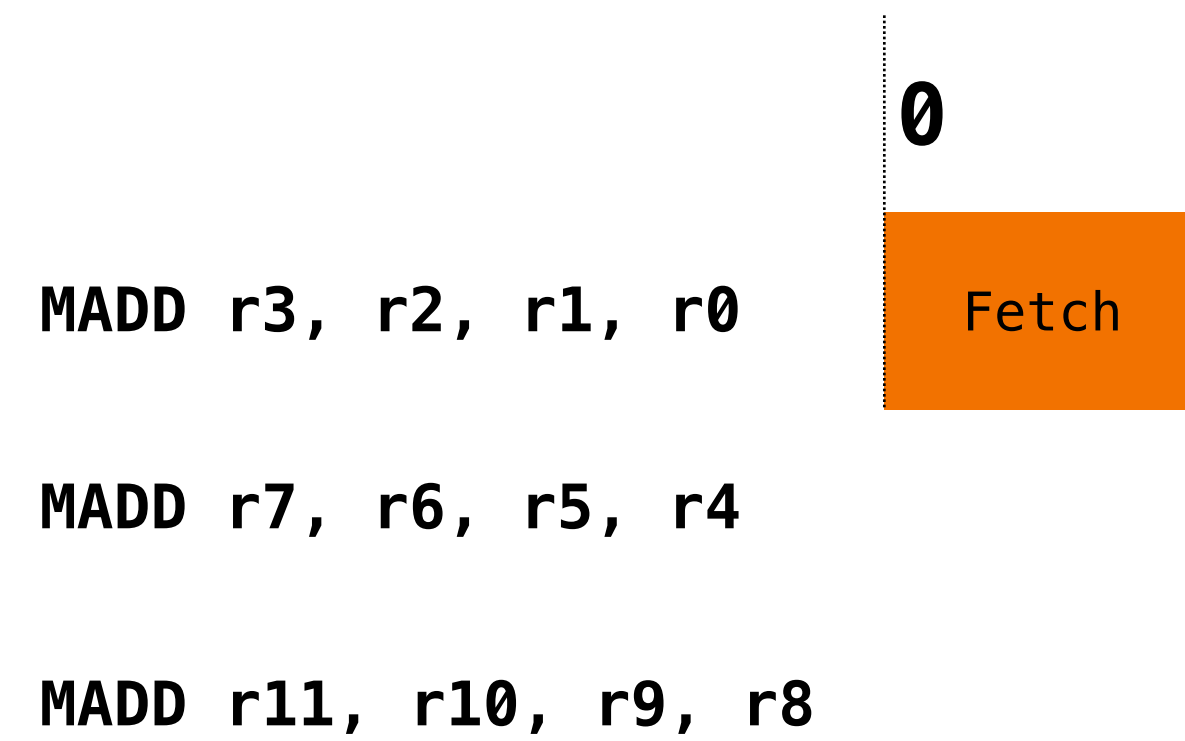
The stage “Execute MADD” can process one instruction at a time

`MADD r3, r2, r1, r0`

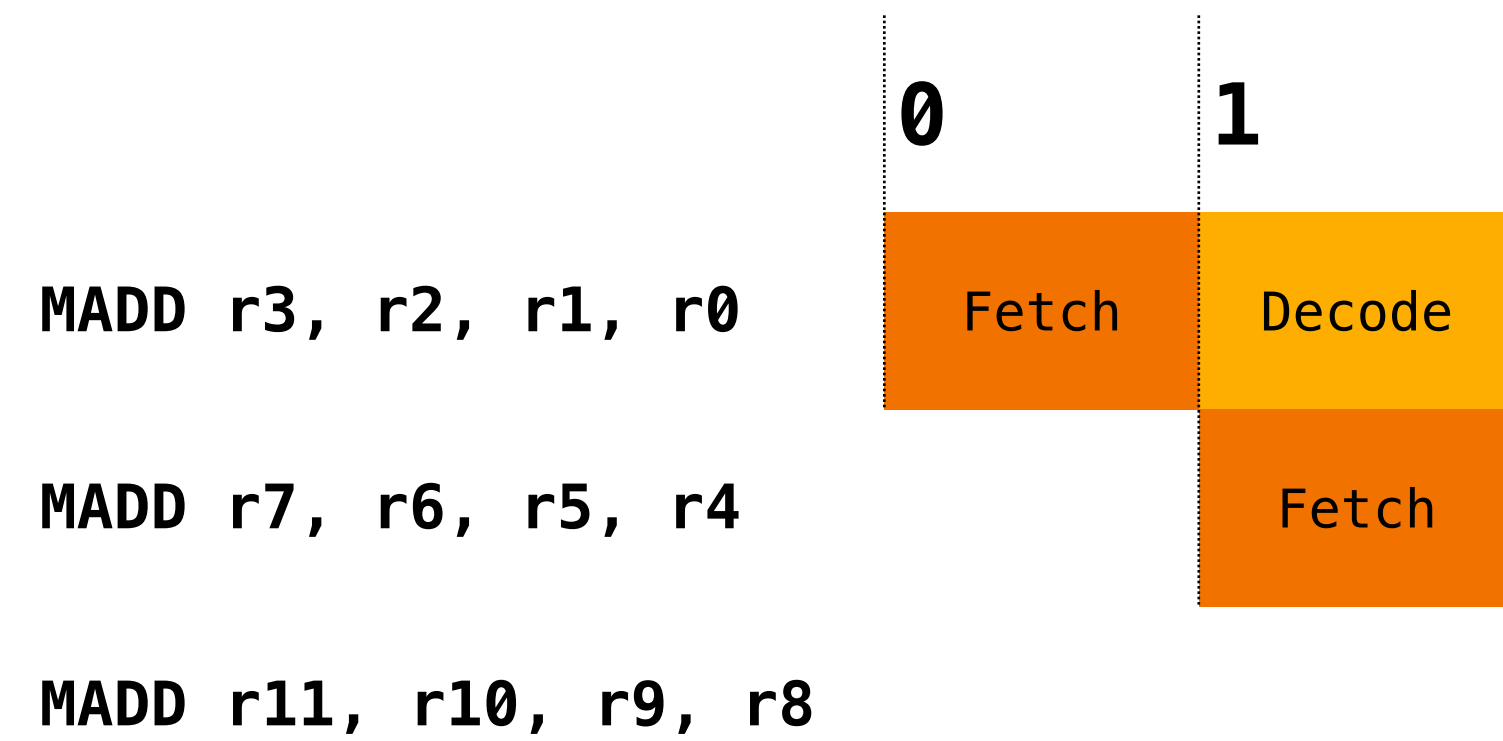
`MADD r7, r6, r5, r4`

`MADD r11, r10, r9, r8`

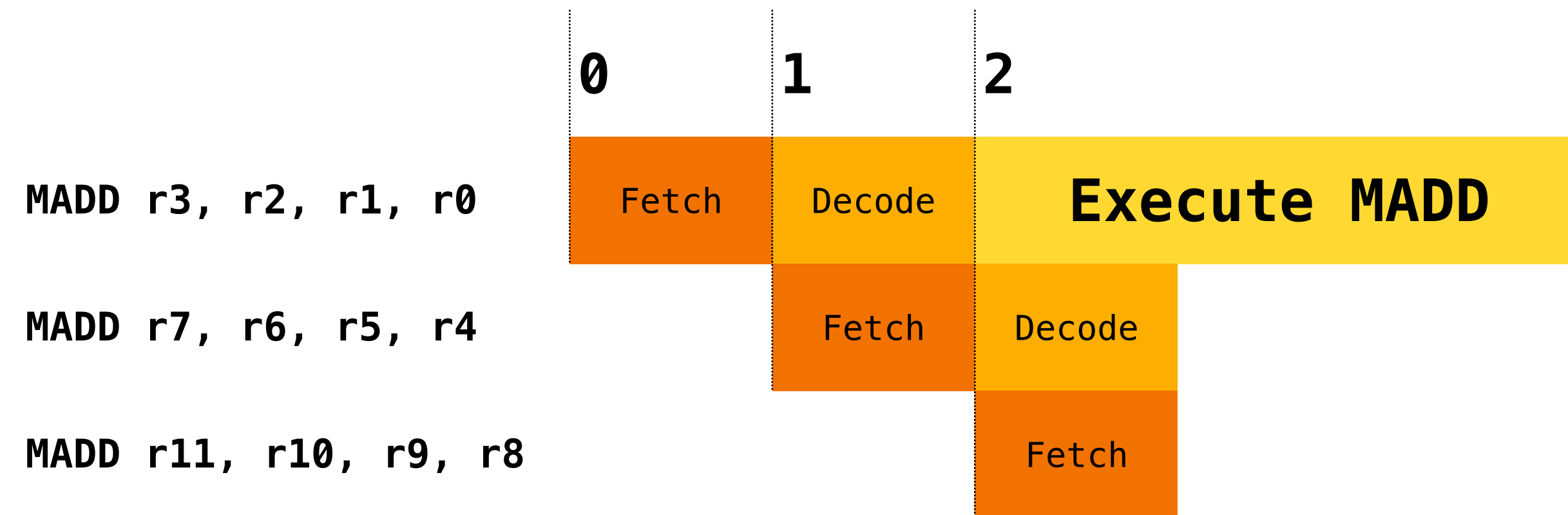
The stage “Execute MADD” can process one instruction at a time



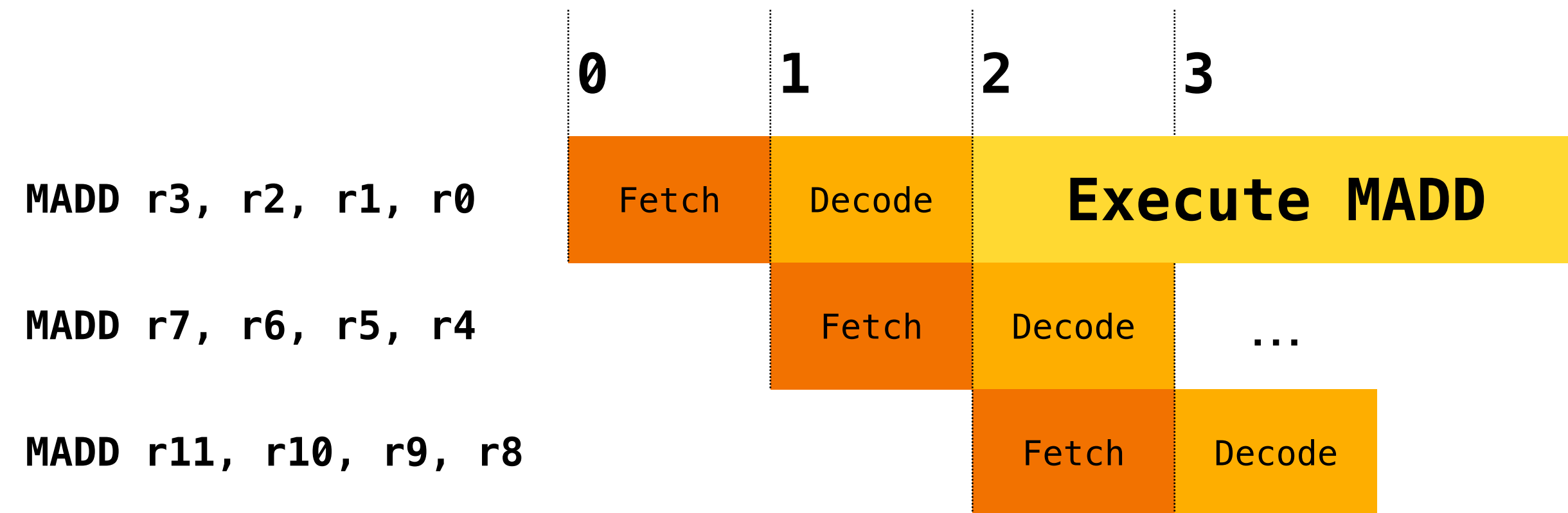
The stage “Execute MADD” can process one instruction at a time



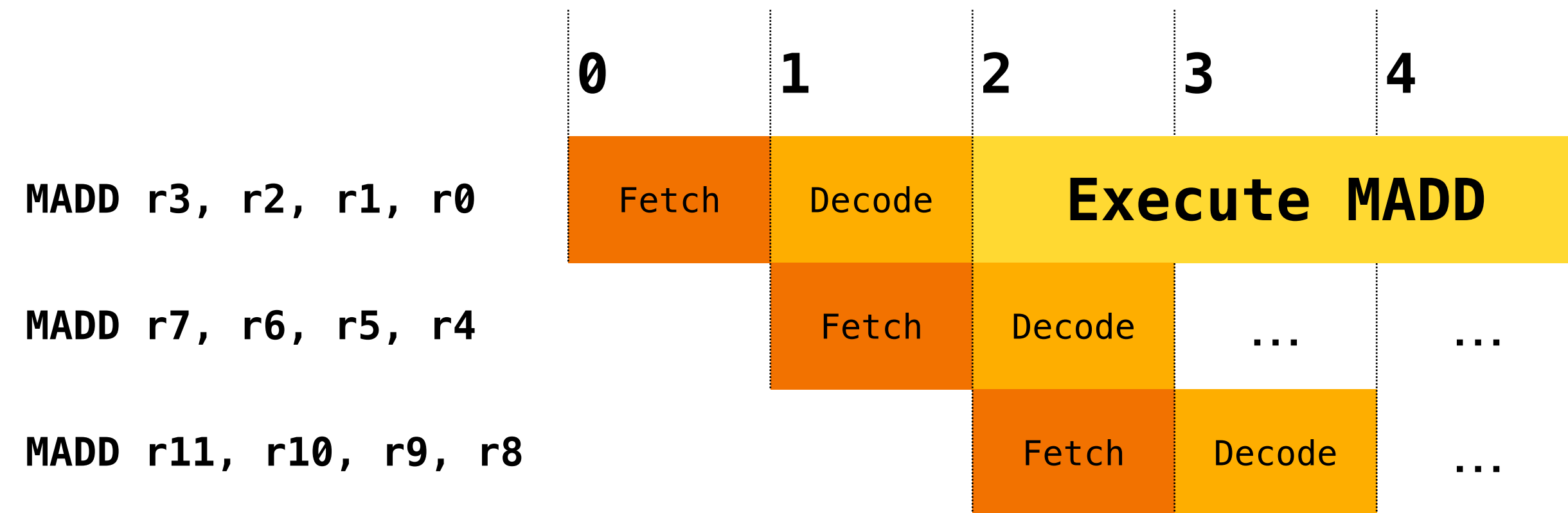
The stage “Execute MADD” can process one instruction at a time



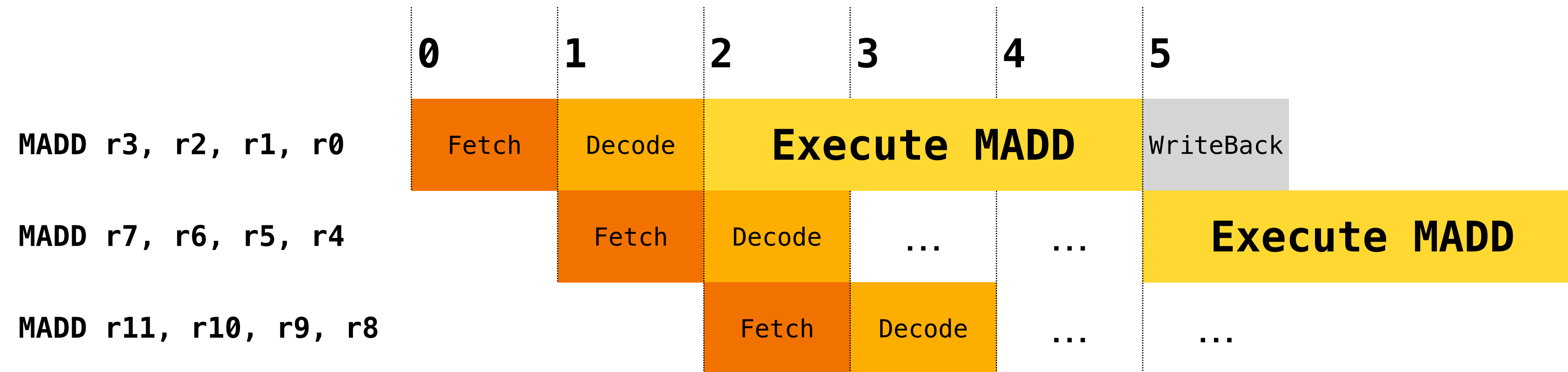
The stage “Execute MADD” can process one instruction at a time



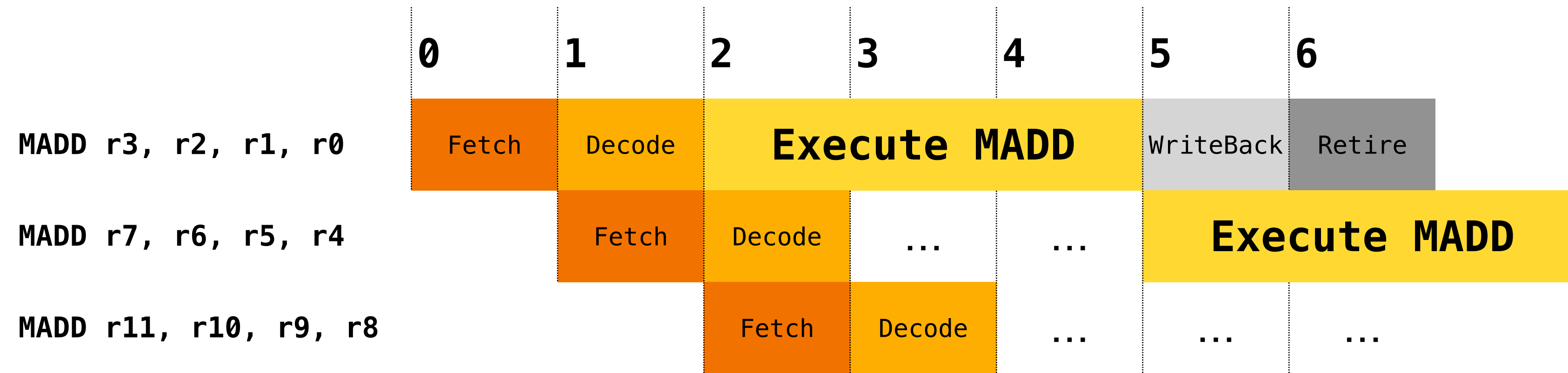
The stage “Execute MADD” can process one instruction at a time



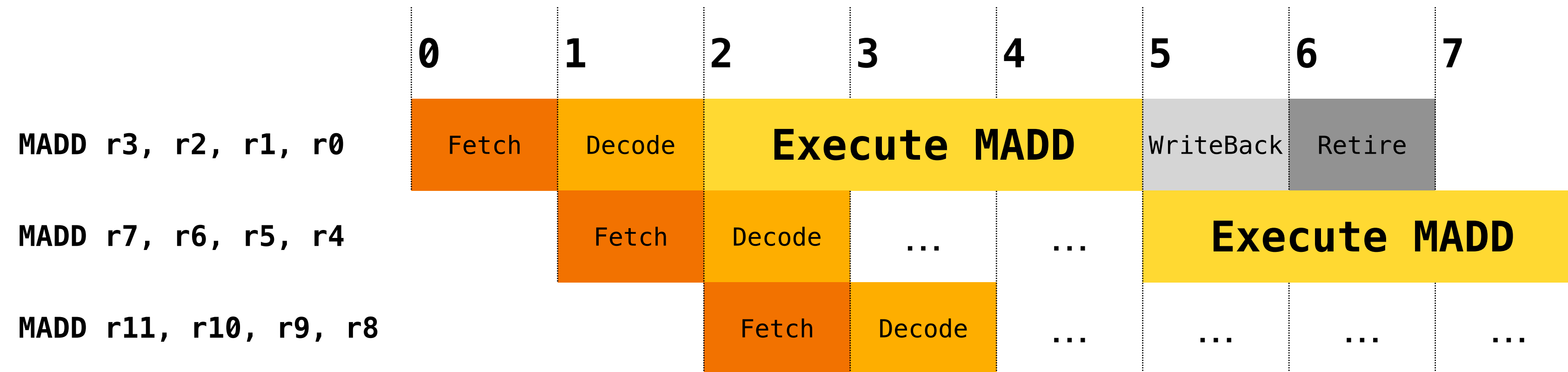
The stage “Execute MADD” can process one instruction at a time



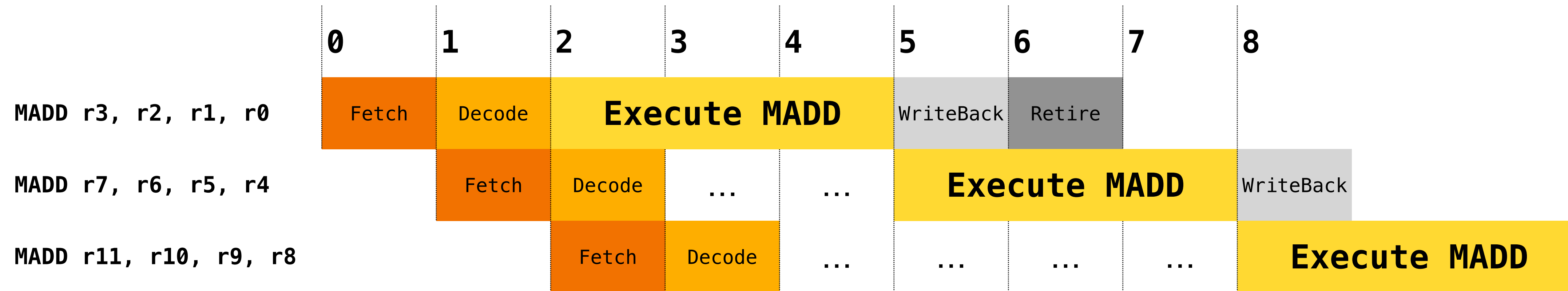
The stage “Execute MADD” can process one instruction at a time



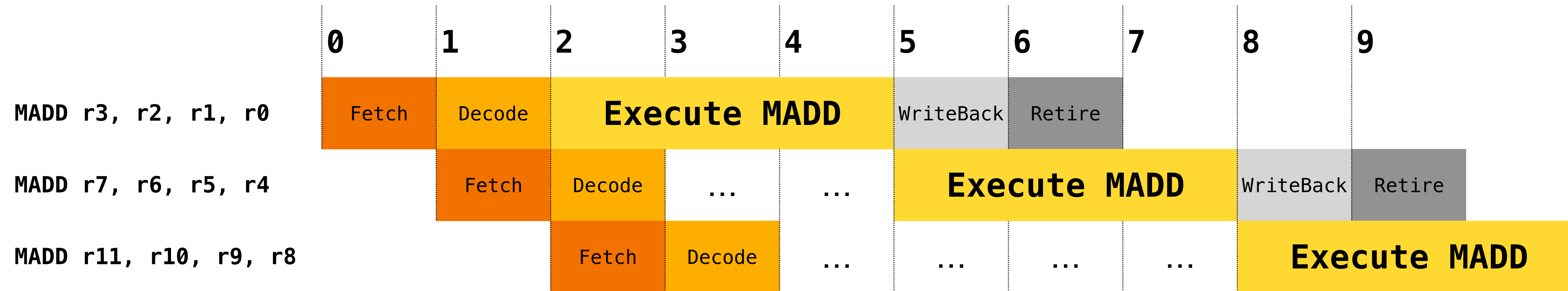
The stage “Execute MADD” can process one instruction at a time



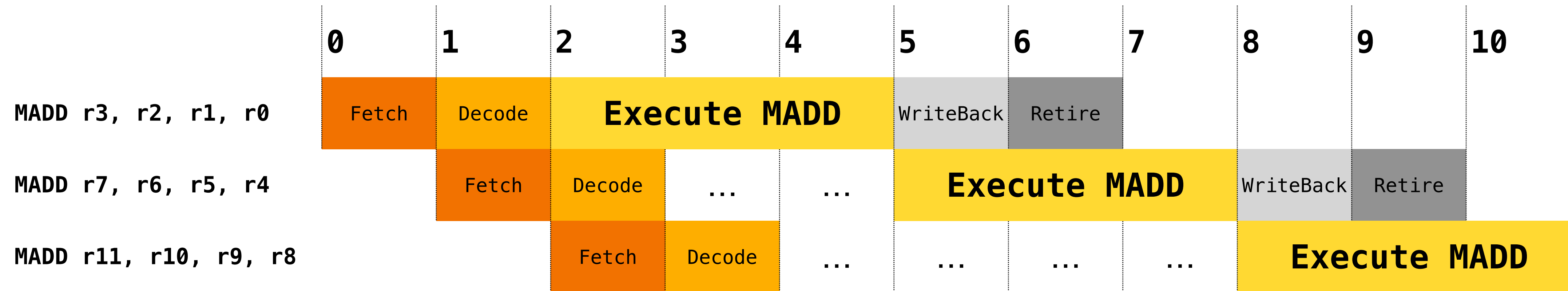
The stage “Execute MADD” can process one instruction at a time



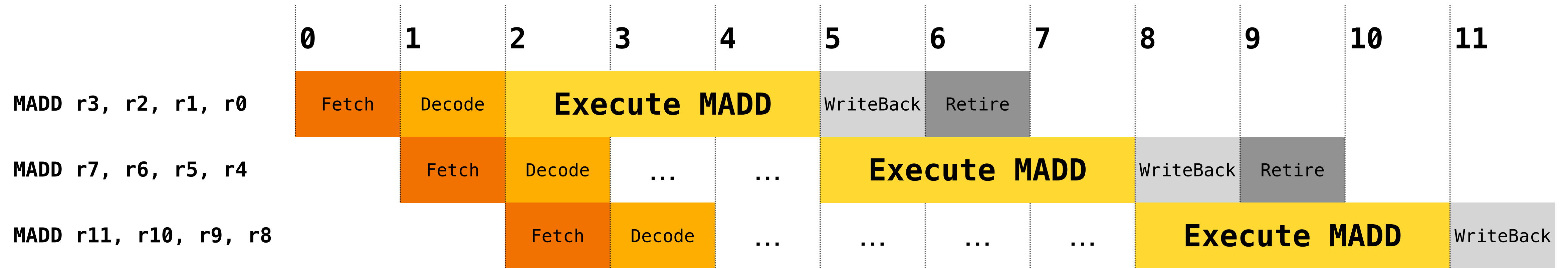
The stage “Execute MADD” can process one instruction at a time



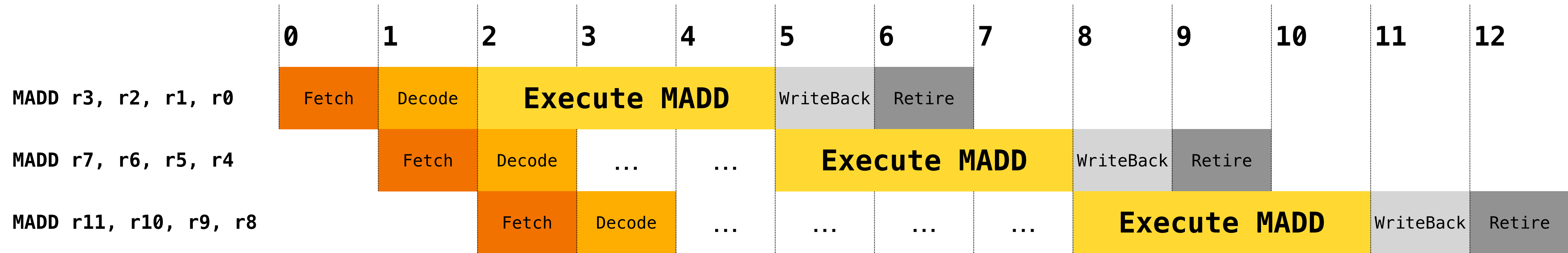
The stage “Execute MADD” can process one instruction at a time



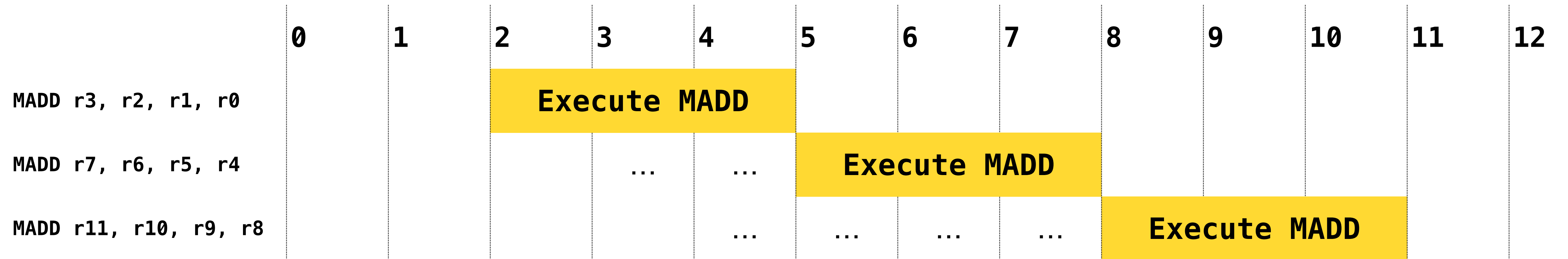
The stage “Execute MADD” can process one instruction at a time



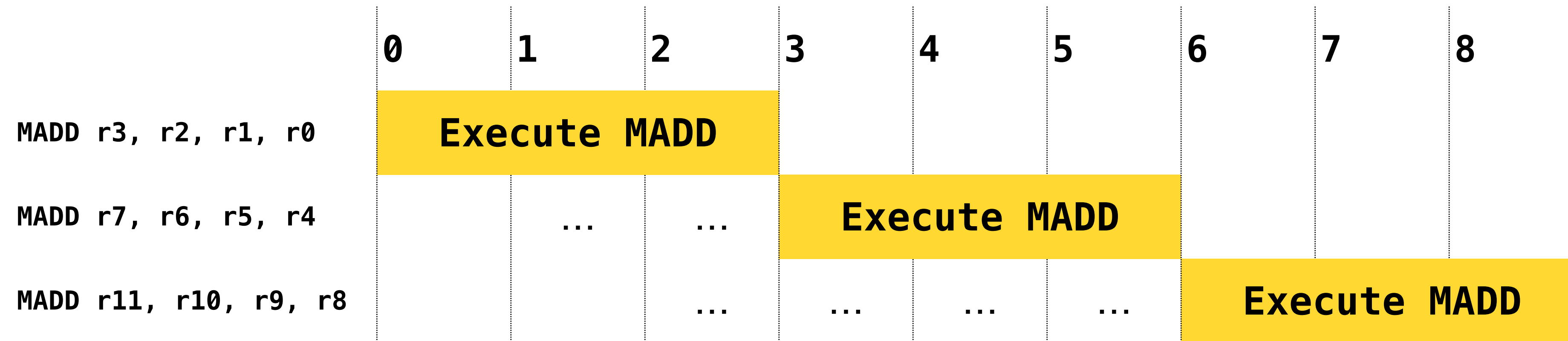
The stage “Execute MADD” can process one instruction at a time



Focus on the instruction-specific resources



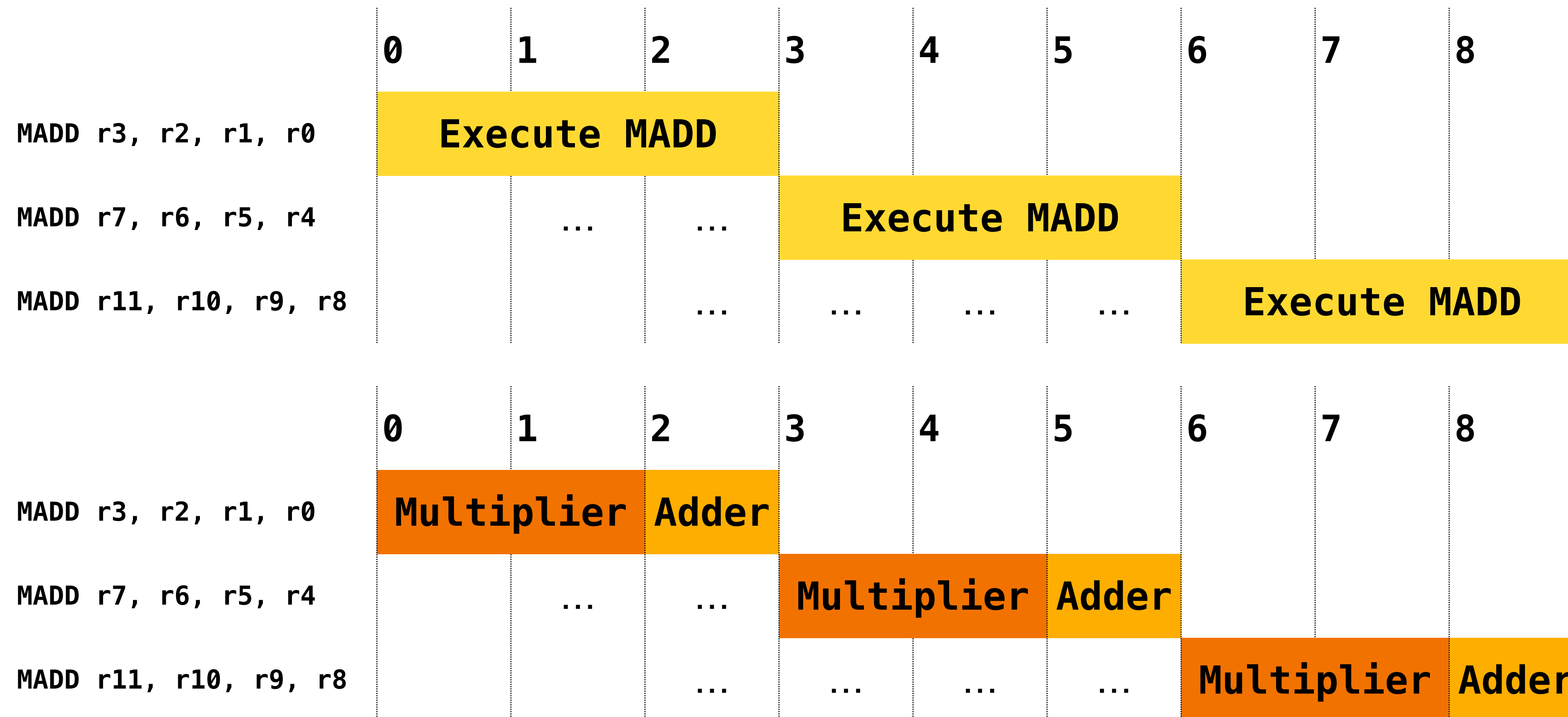
Focus on the instruction-specific resources



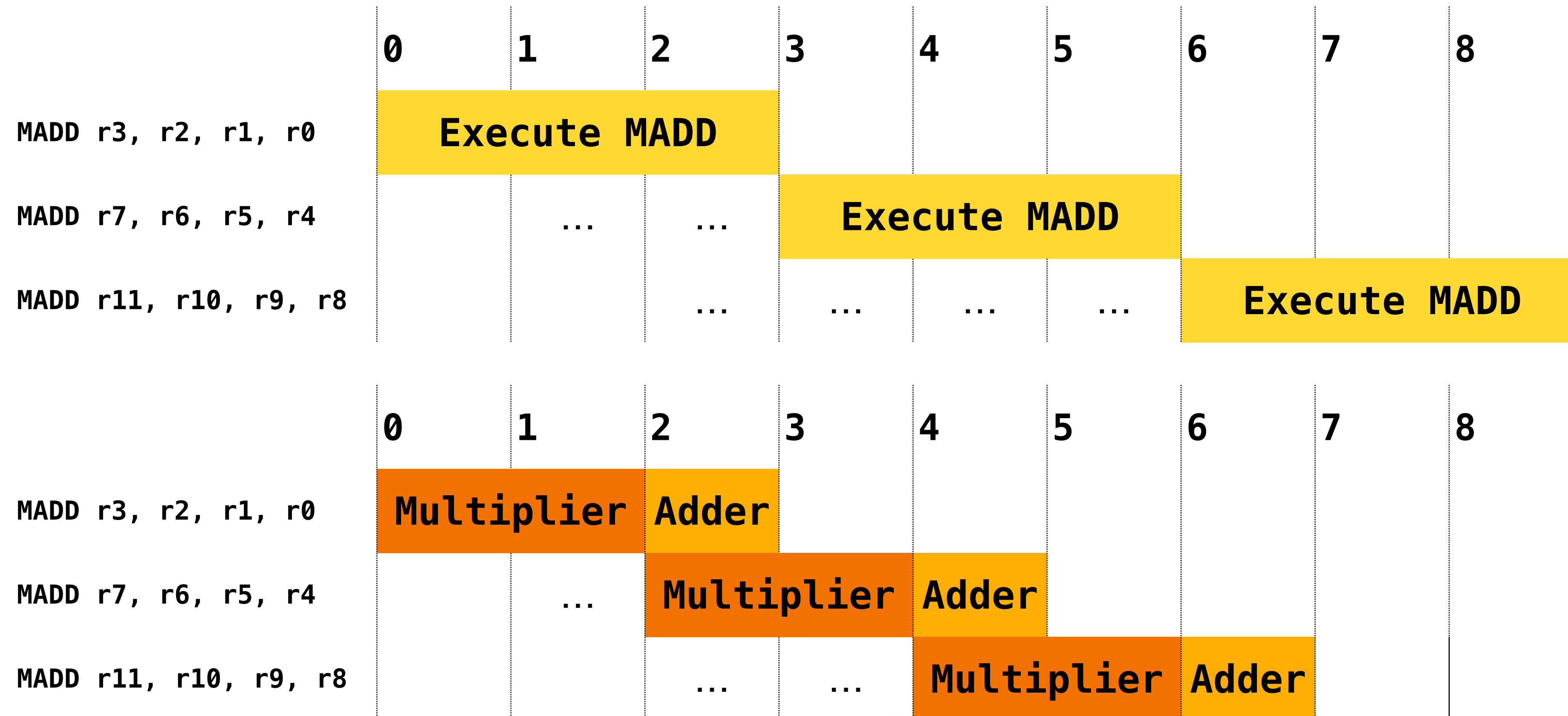
Pipelined execution

Break up the execution in separate stages

Hardware feature



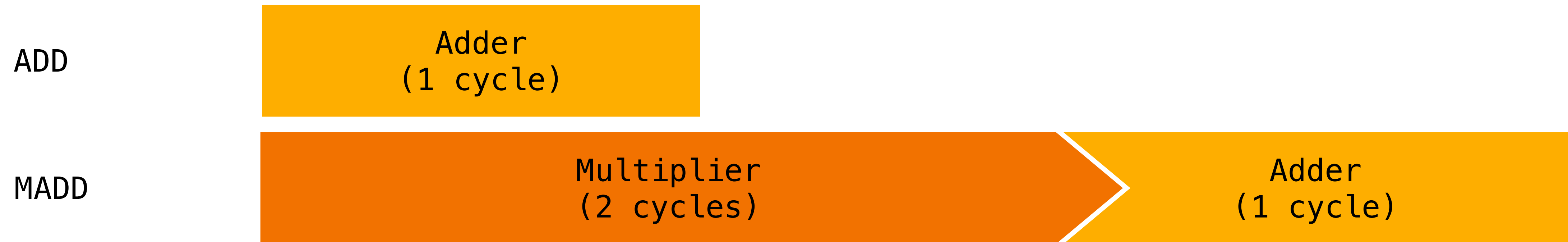
Pipelined resources: faster execution!



What happens when pipeline execution shares functional units?

Reminder

Focus on the execution units



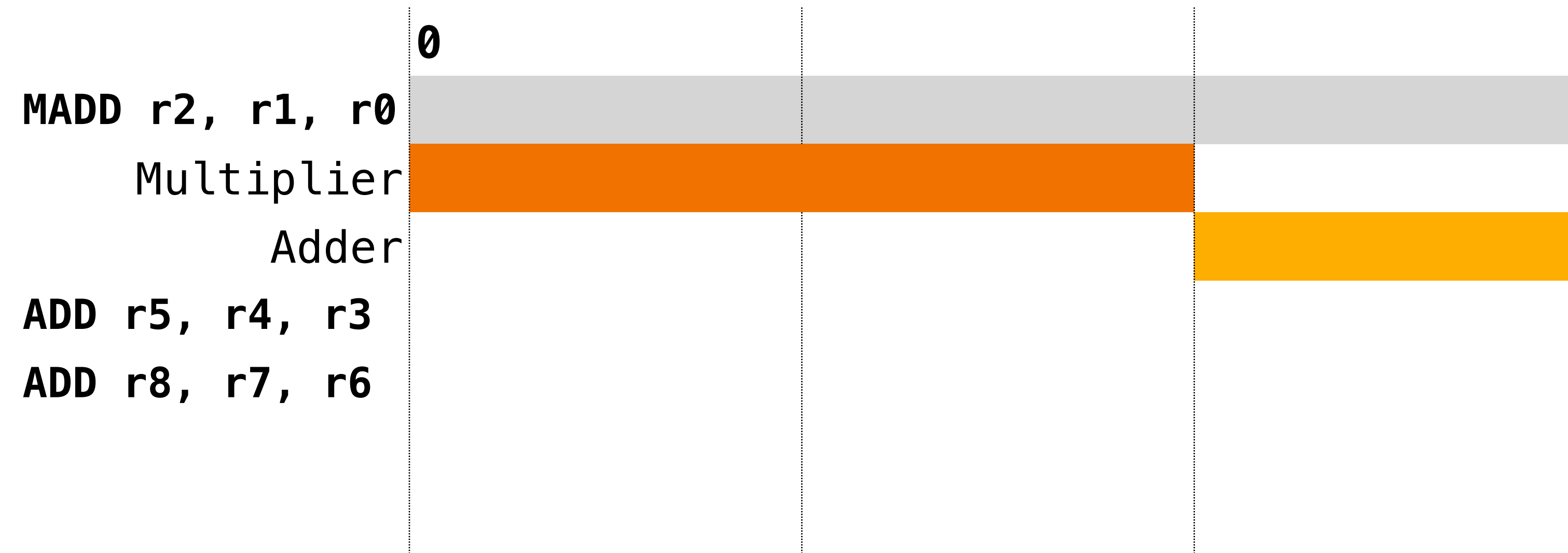
Execution with a shared Adder

MADD r2, r1, r0

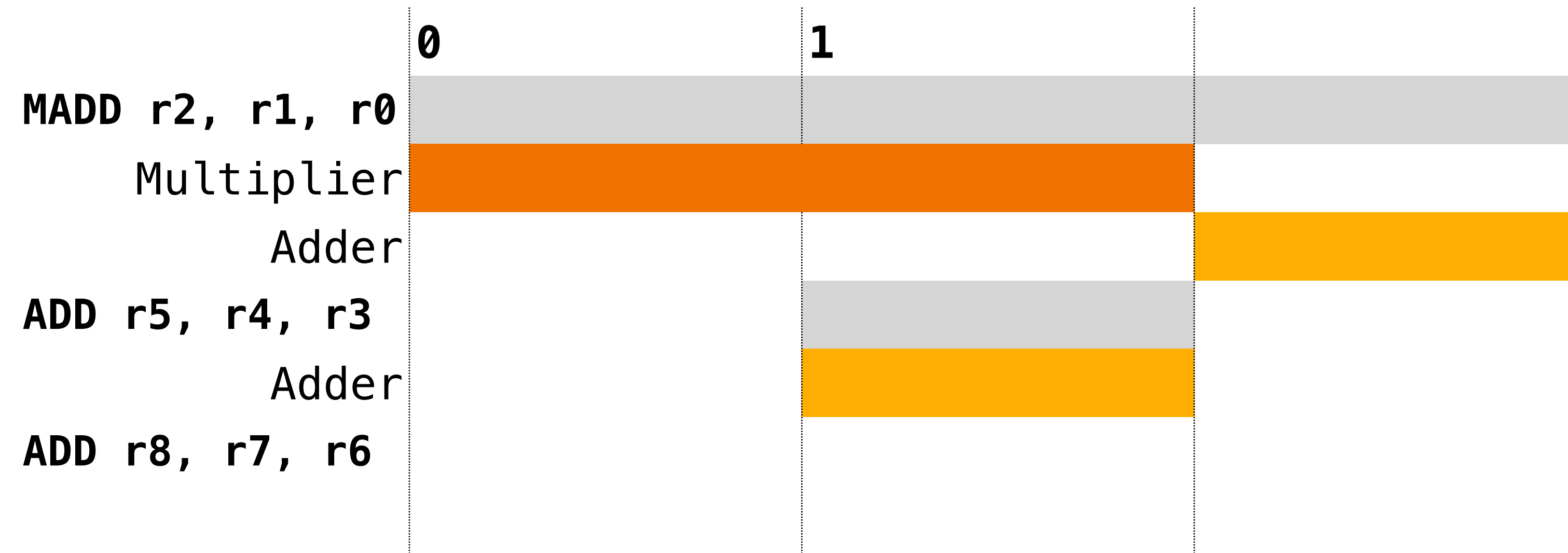
ADD r5, r4, r3

ADD r8, r7, r6

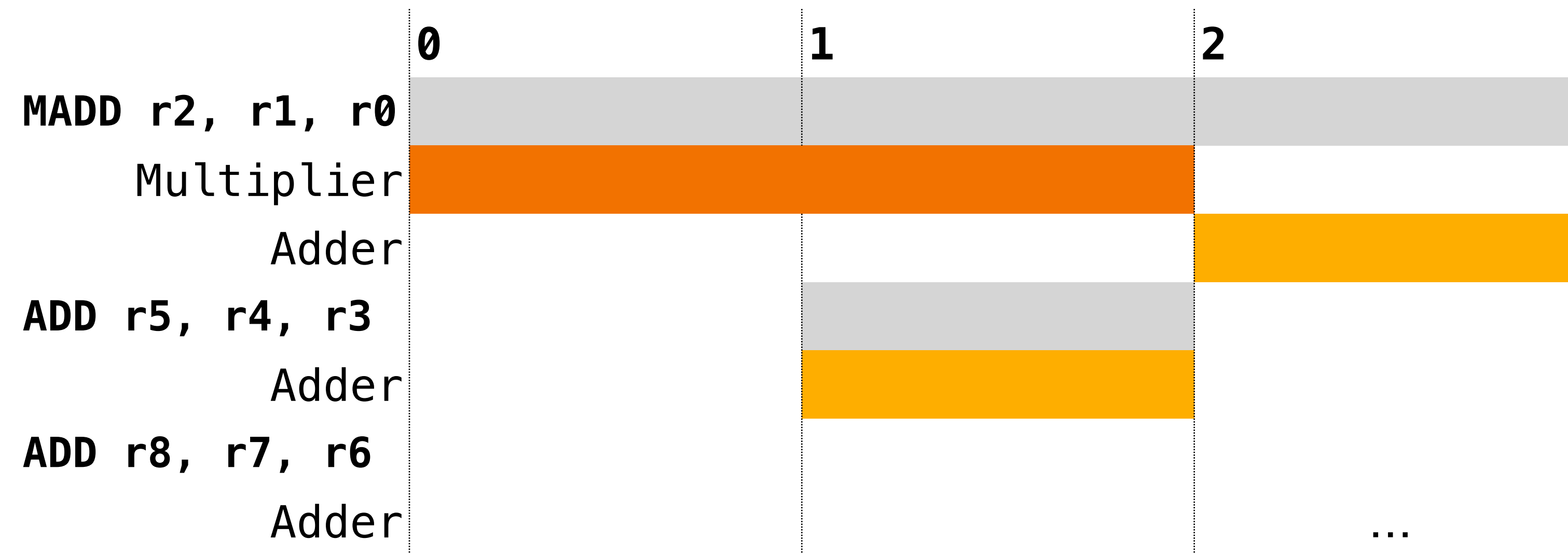
Execution with a shared Adder



Execution with a shared Adder

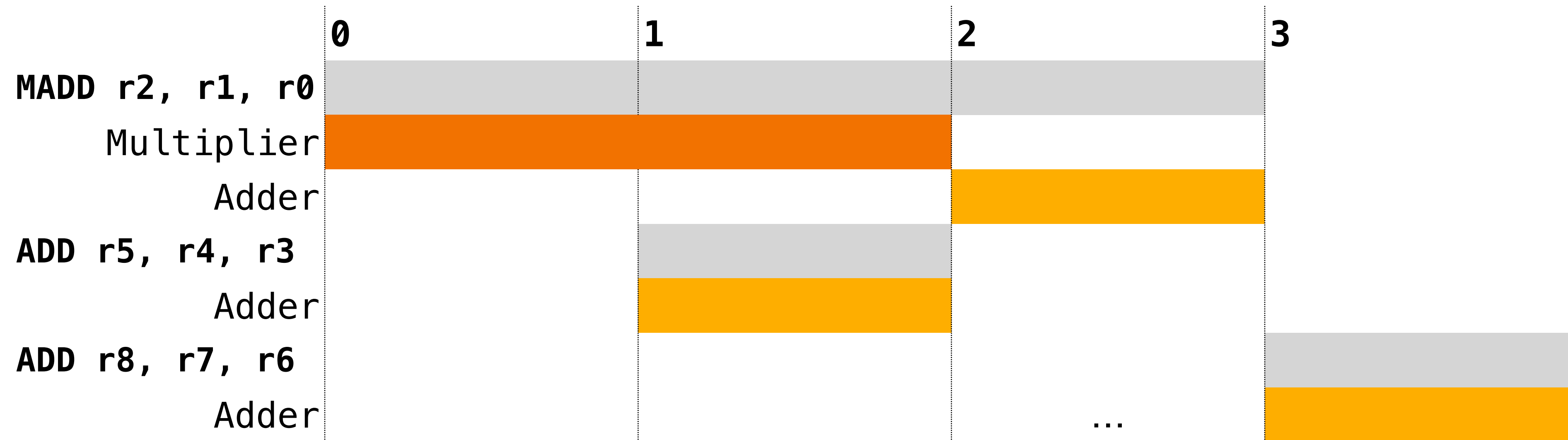


Execution with a shared Adder

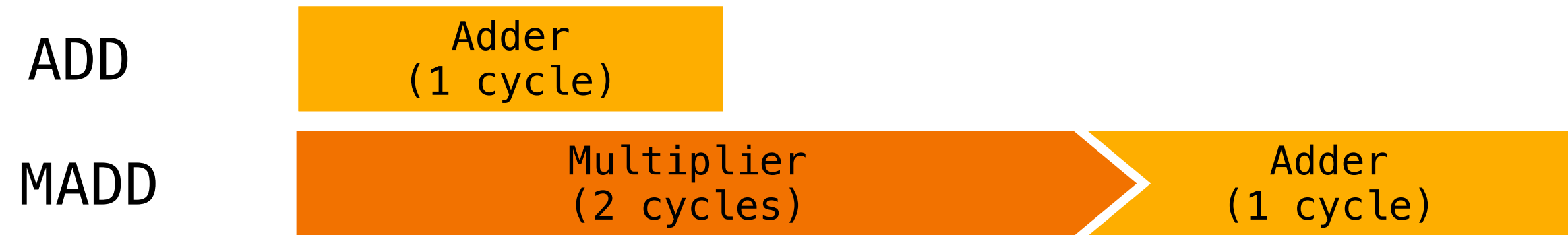


Execution with a shared Adder

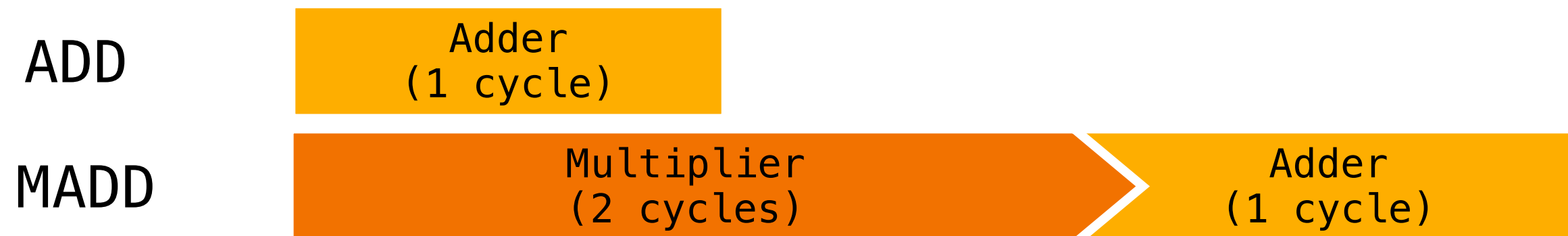
Postpone execution (stall)



LLVM's representation of resources



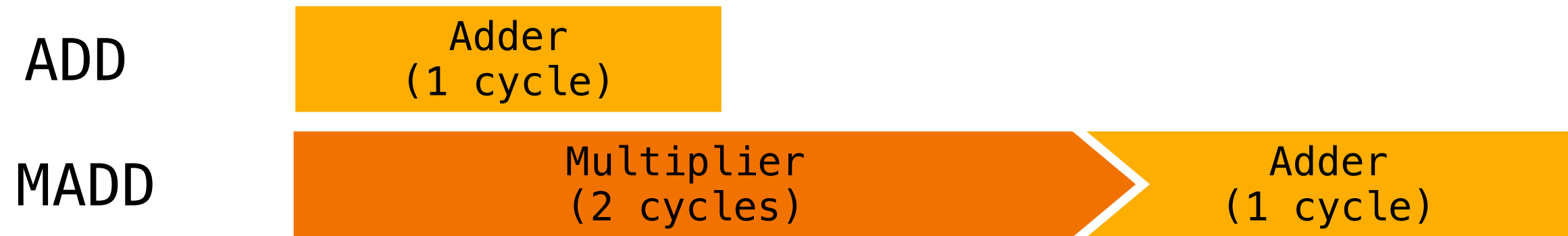
LLVM's representation of resources



TableGen description

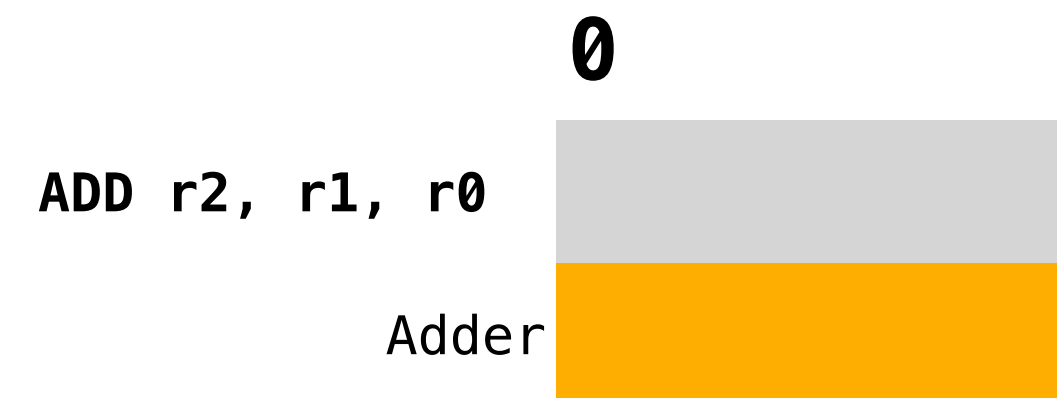
```
def : WriteRes<WriteADD, [Adder]> {  
    let ResourceCycles = [ 1];  
}
```


LLVM's representation of resources

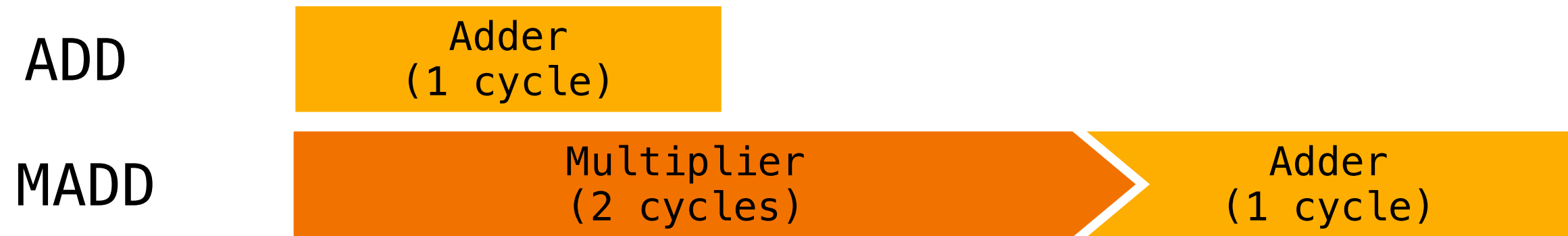


TableGen description

```
def : WriteRes<WriteADD, [Adder]> {  
  let ResourceCycles = [ 1];  
}
```

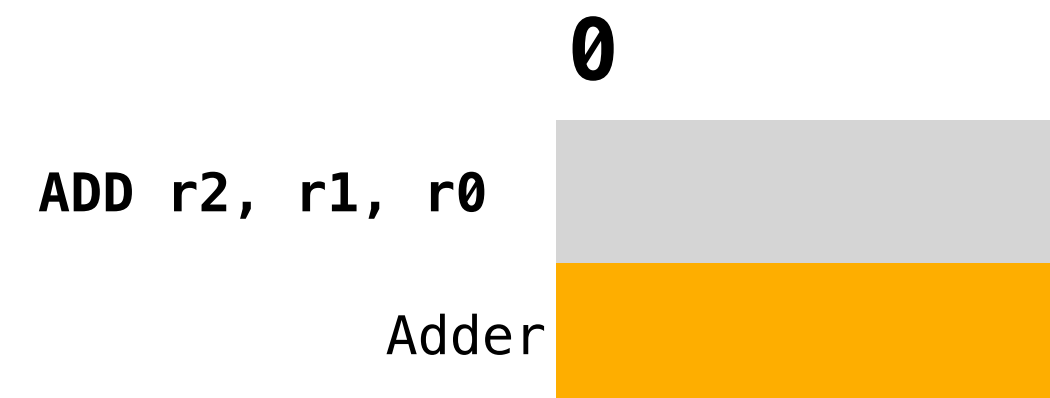


LLVM's representation of resources

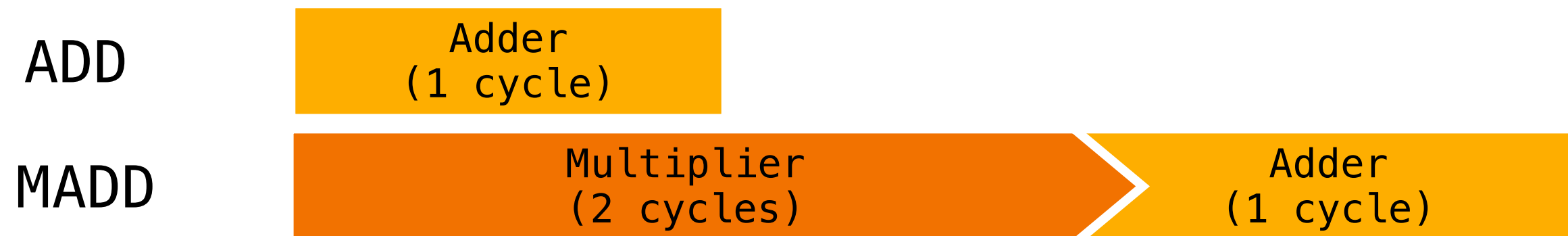


TableGen description

```
def : WriteRes<WriteADD, [Adder]> {  
  let ResourceCycles = [ 1];  
}  
def : WriteRes<WriteMADD, [Multiplier, Adder]> {  
  let ResourceCycles = [ 2, 3];  
}
```

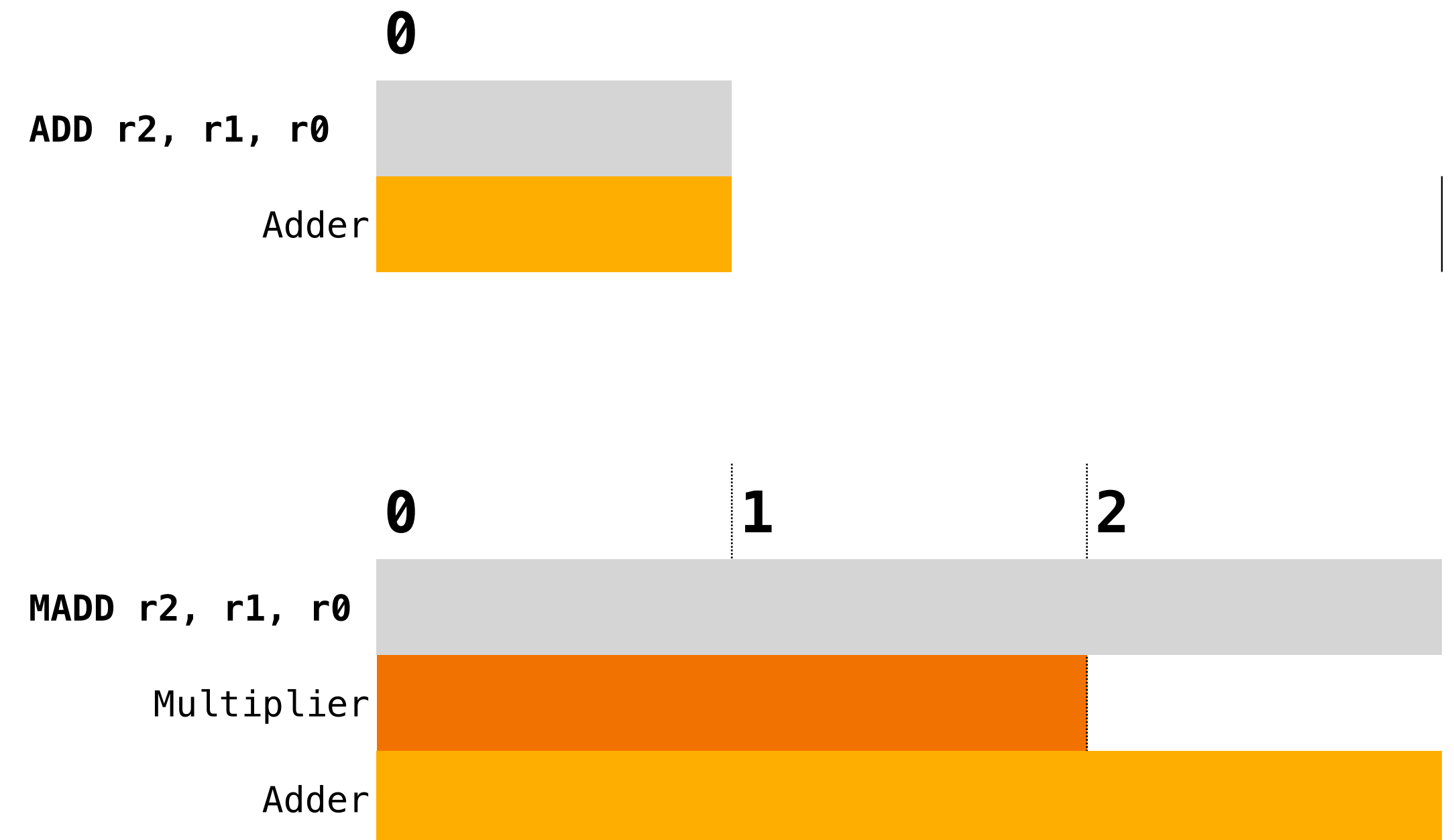


LLVM's representation of resources

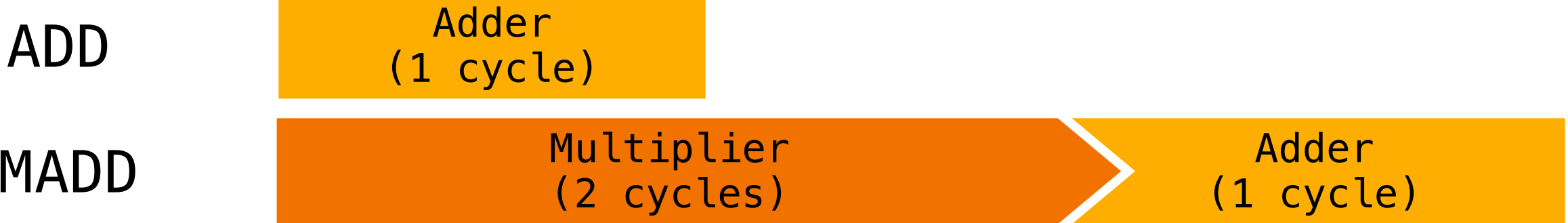


TableGen description

```
def : WriteRes<WriteADD, [Adder]> {  
  let ResourceCycles = [ 1];  
}  
def : WriteRes<WriteMADD, [Multiplier, Adder]> {  
  let ResourceCycles = [ 2, 3];  
}
```



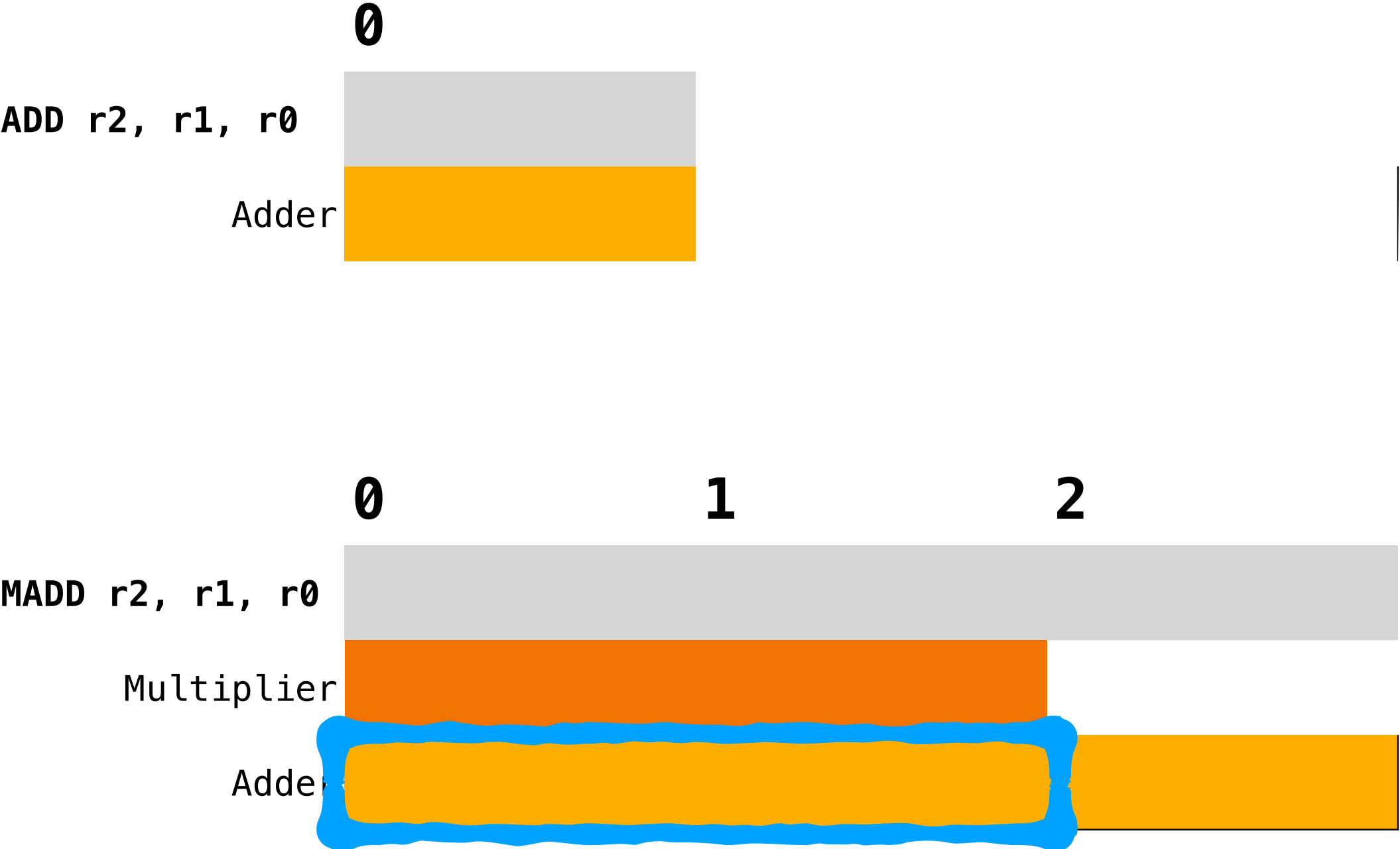
LLVM's representation of resources



TableGen description

```
def : WriteRes<WriteADD, [Adder]> {  
  let ResourceCycles = [ 1];  
}  
def : WriteRes<WriteMADD, [Multiplier, Adder]> {  
  let ResourceCycles = [ 2, 3];  
}
```

The Adder resource is overbooked for 2 extra cycles in the **MADD** instruction



**LLVM estimation of execution
with shared resources.**

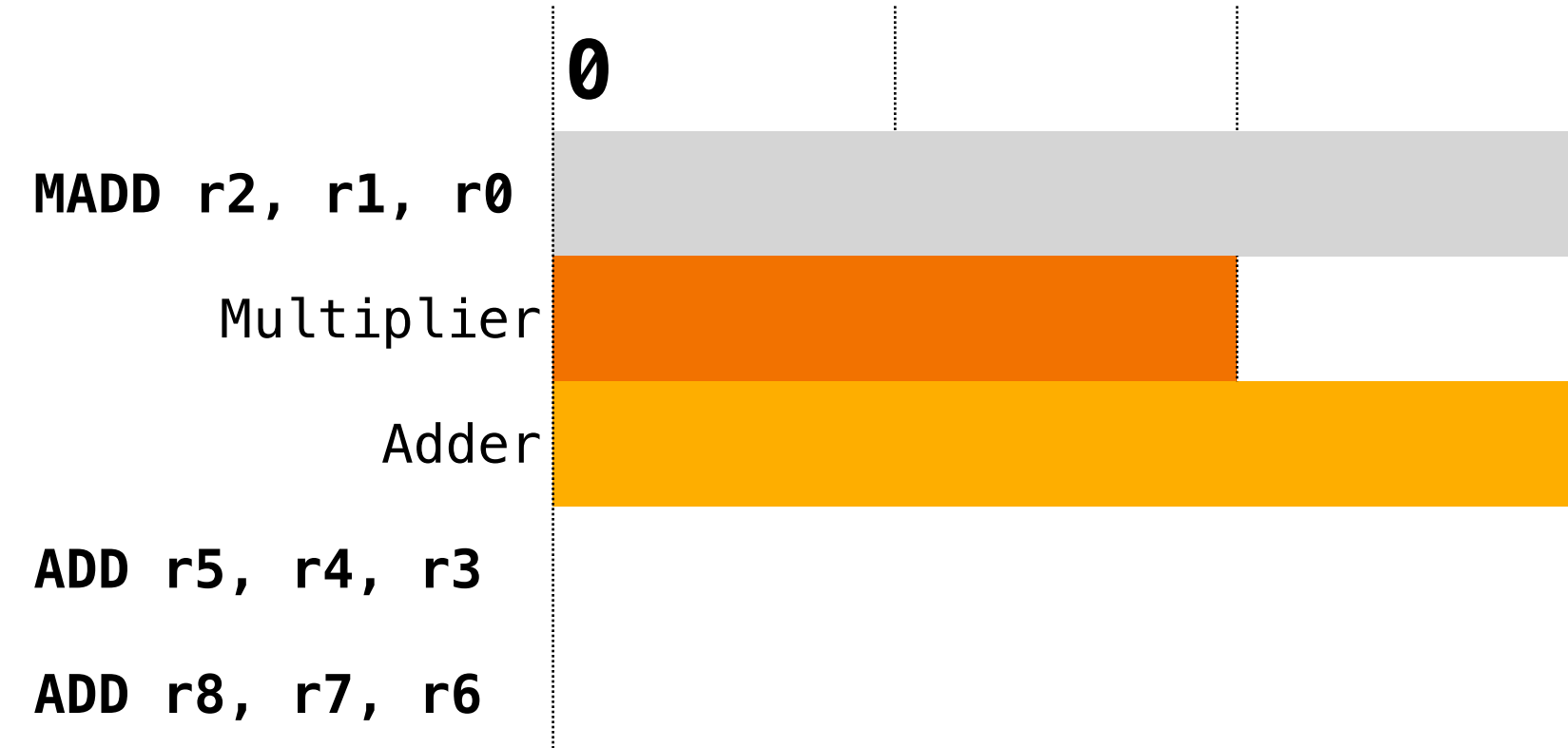
What LLVM estimates

```
MADD r2, r1, r0
```

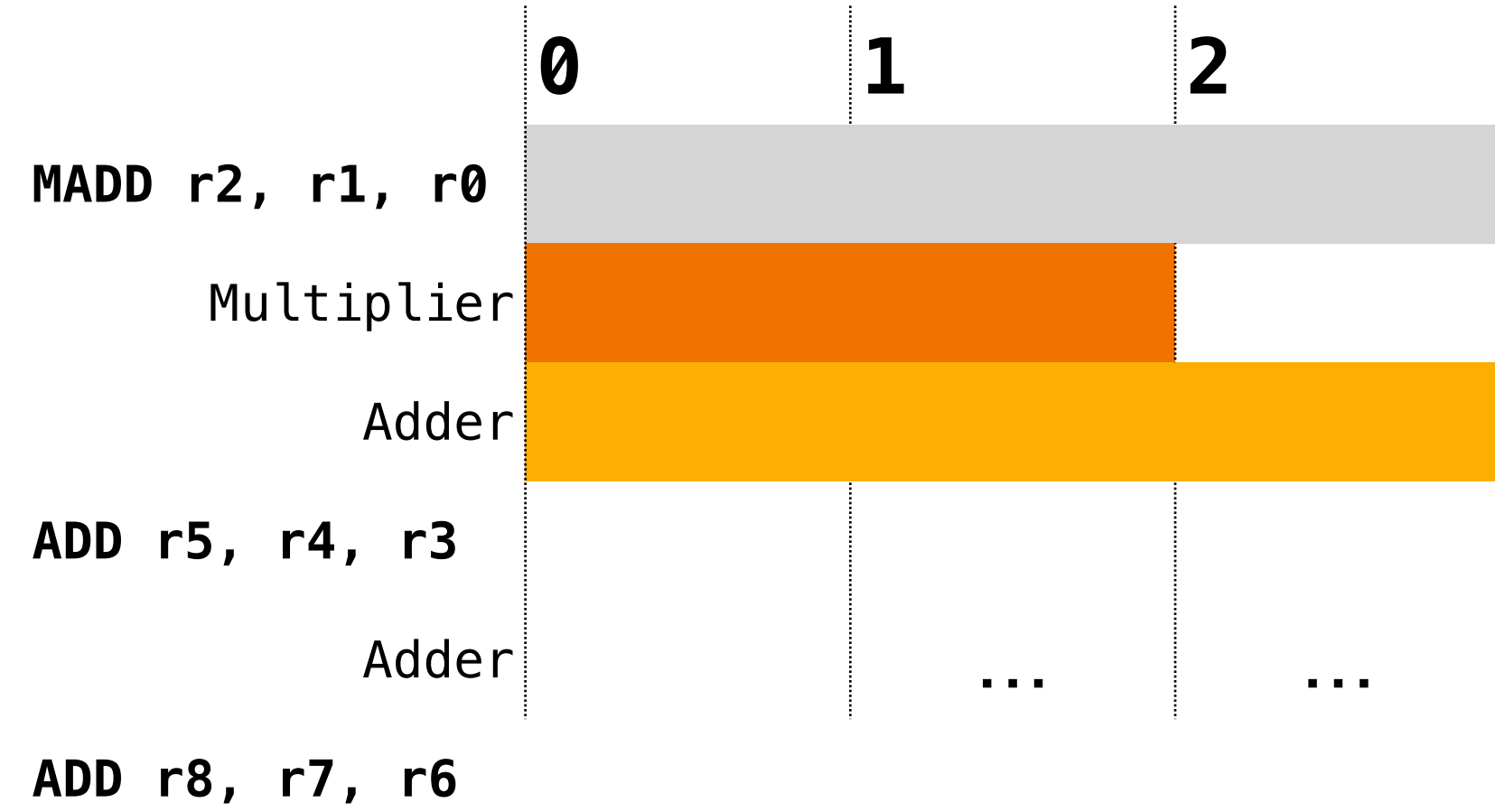
```
ADD r5, r4, r3
```

```
ADD r8, r7, r6
```

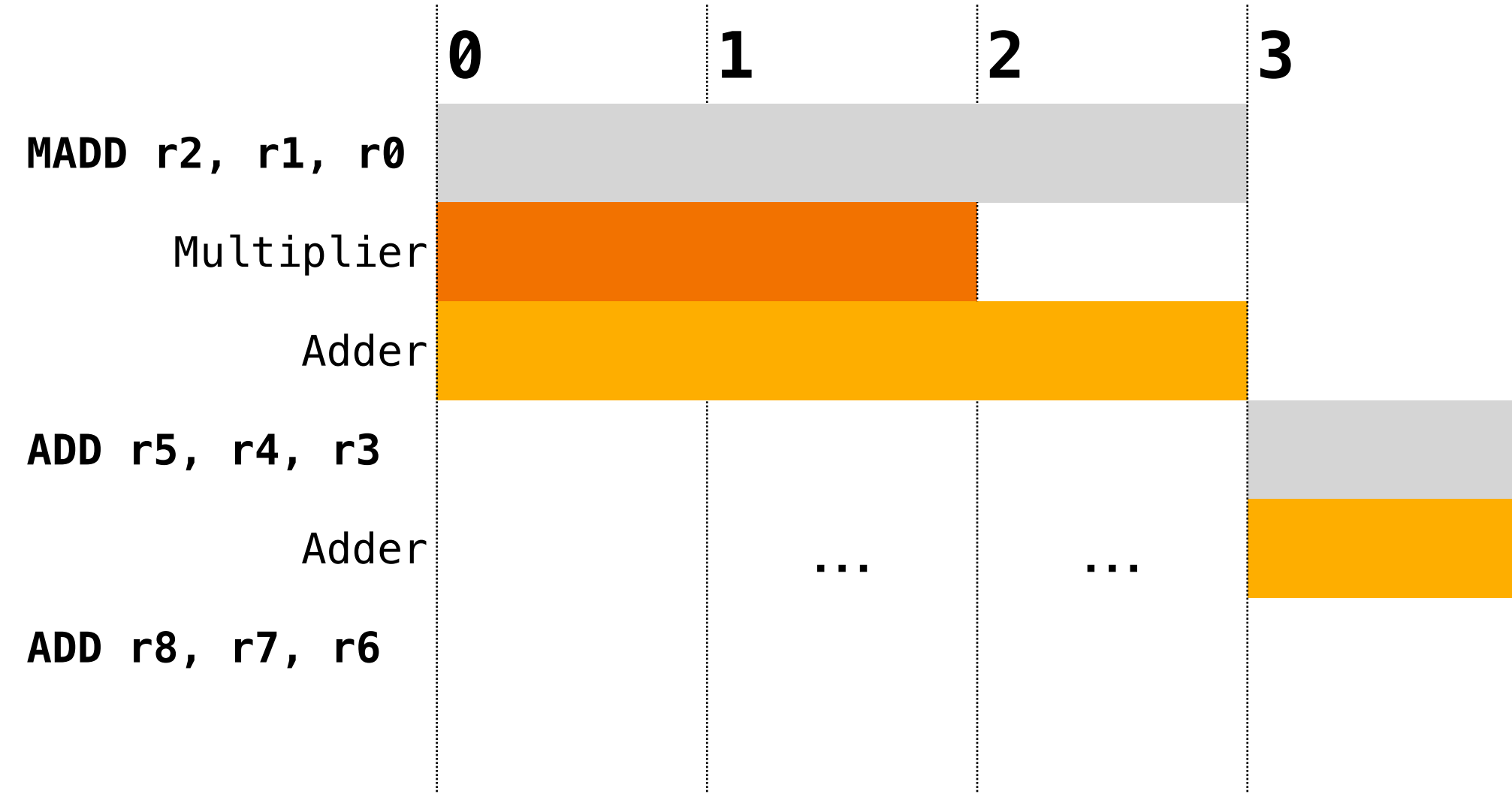
What LLVM estimates



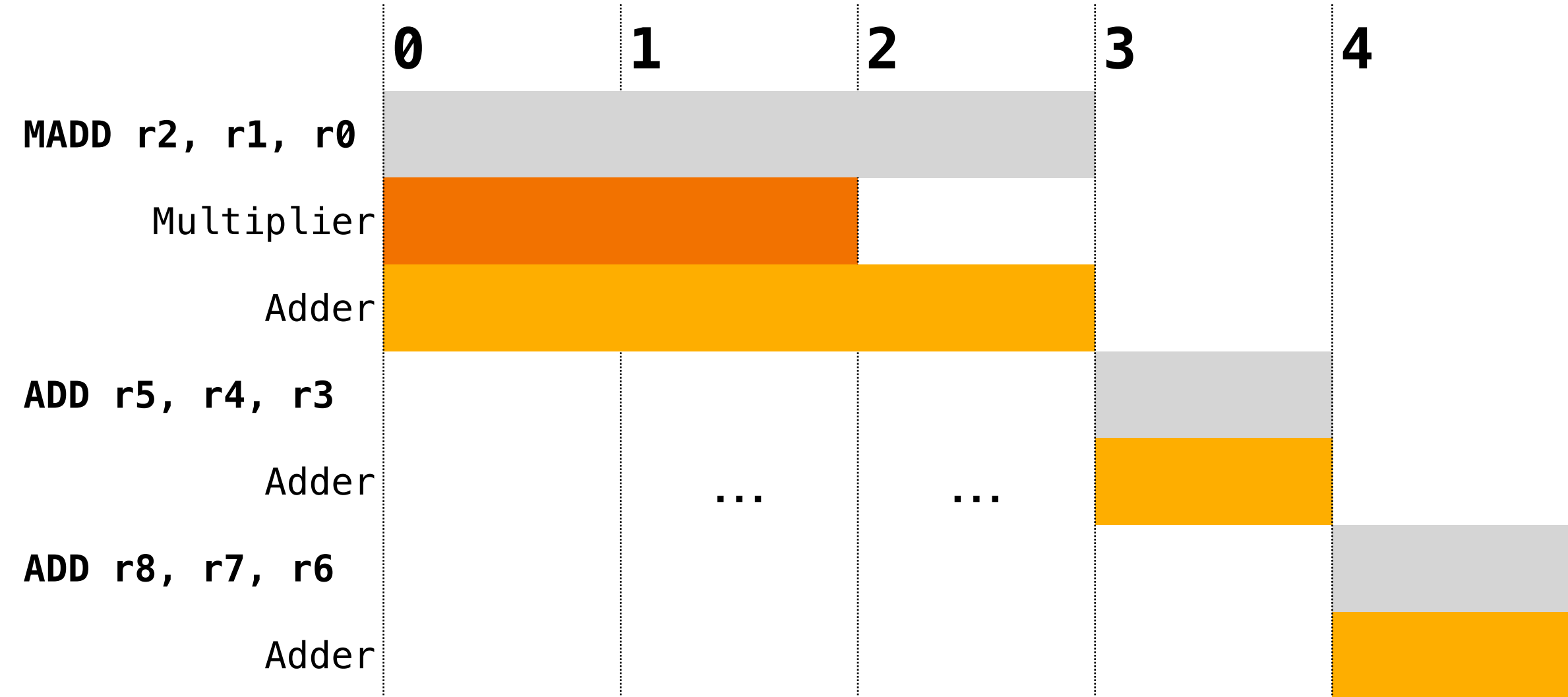
What LLVM estimates



What LLVM estimates

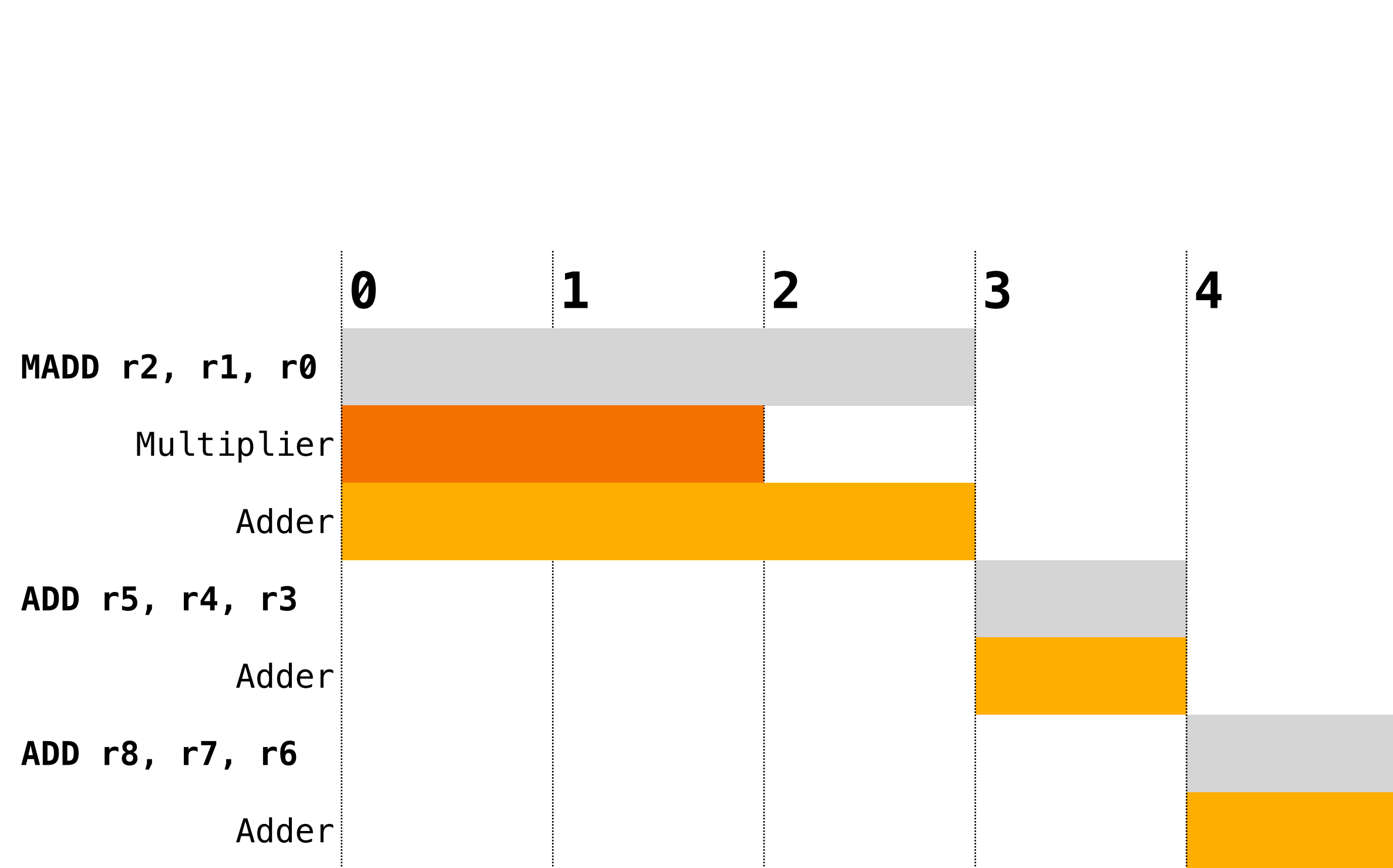


What LLVM estimates

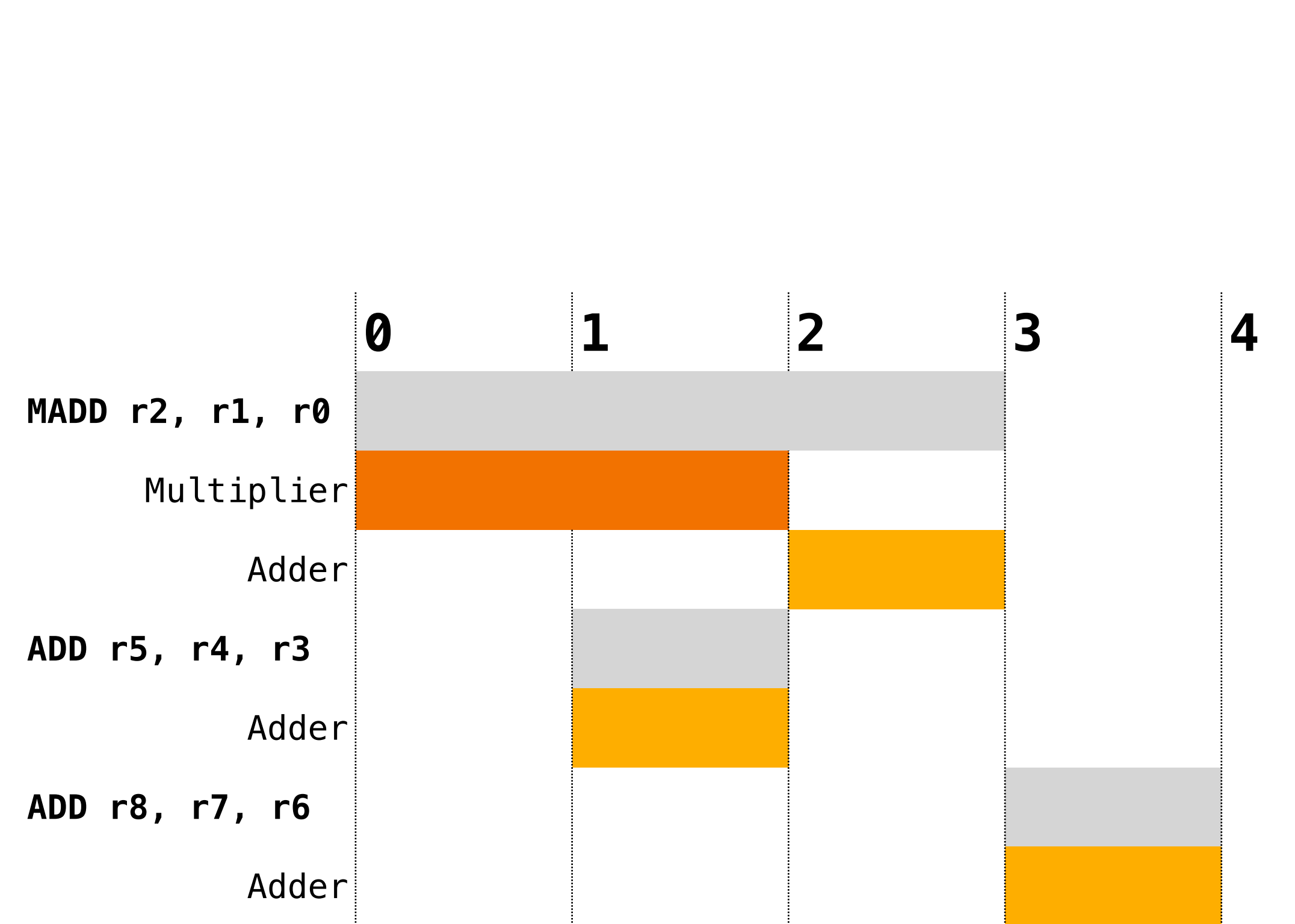


Overbooking of resources leads to longer traces

What LLVM estimates

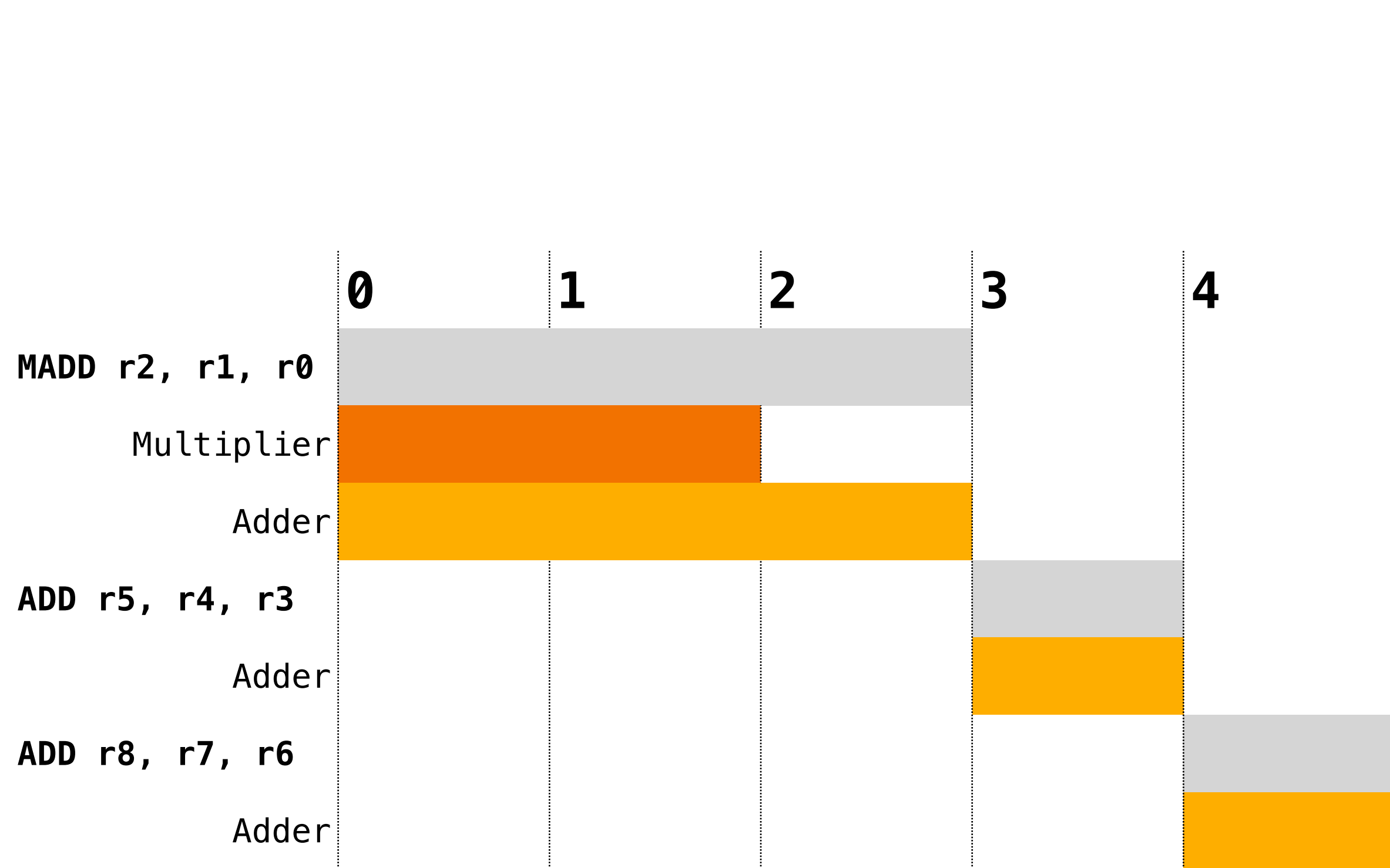


What hardware does

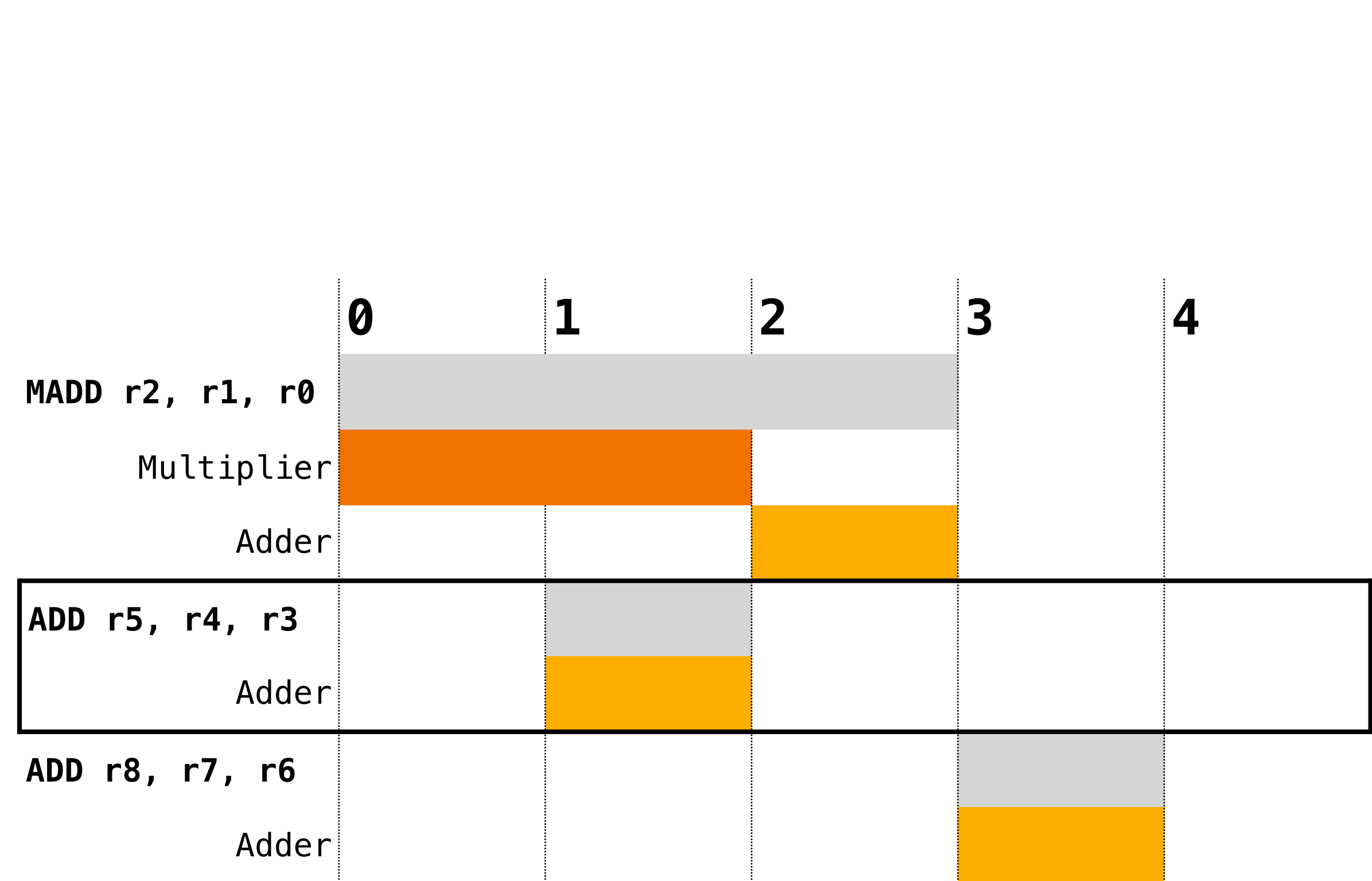


Overbooking of resources leads to longer traces

What LLVM estimates



What hardware does

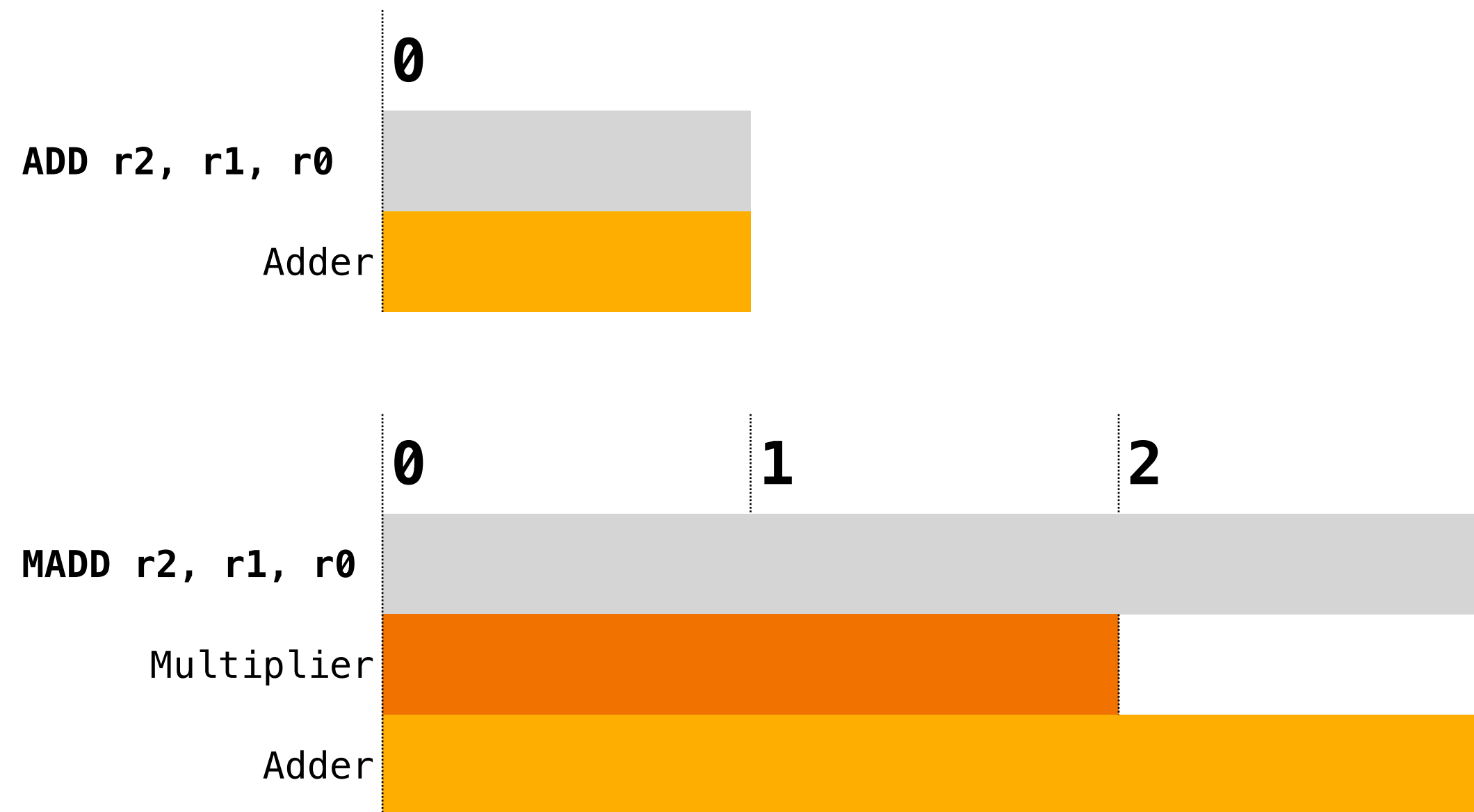


Resource Segments

Fine grain resolution of resource usage

ResourceSegments

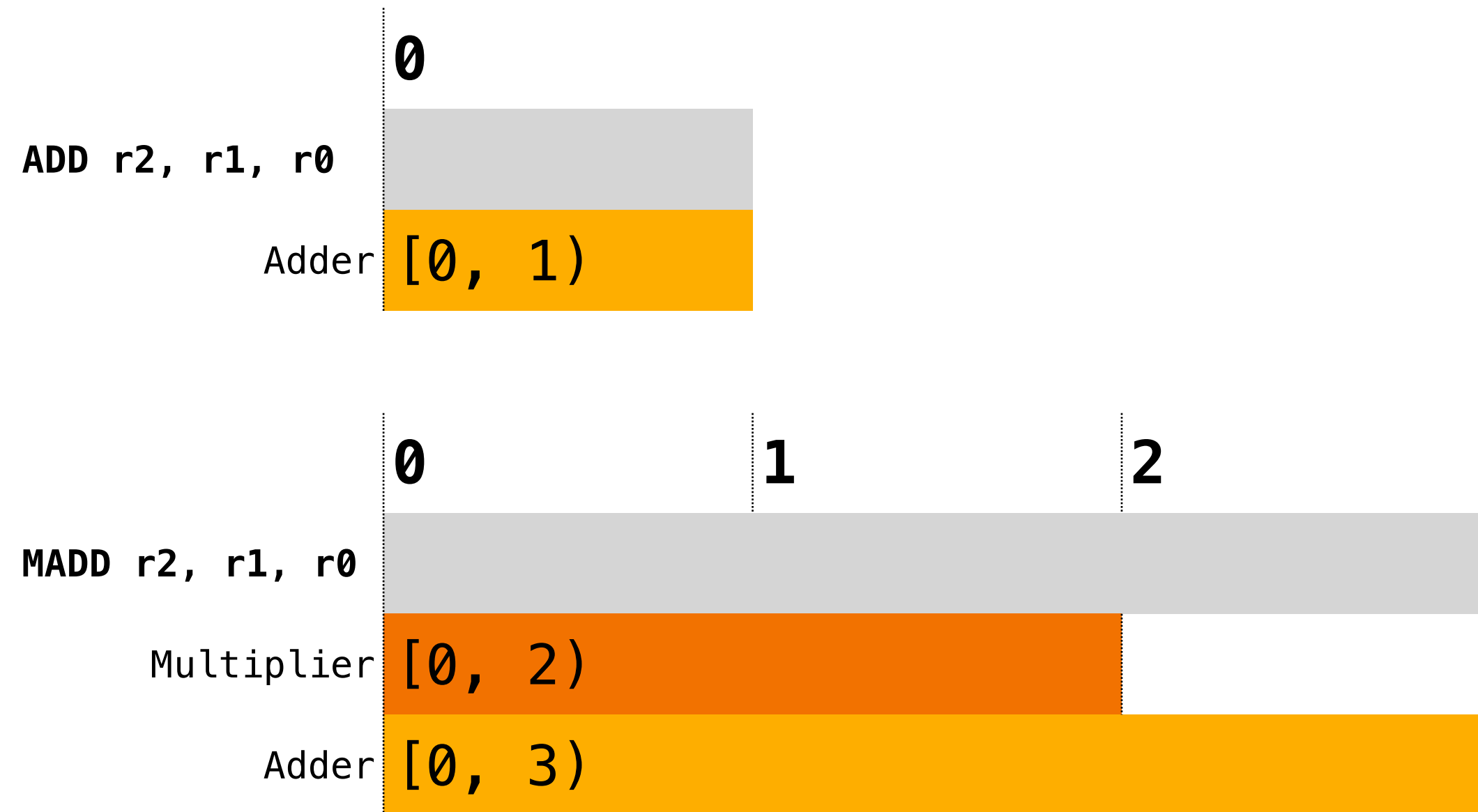
Thinking in terms of intervals, open on the right: [A, B)



```
def : WriteRes<WriteADD, [Adder]> {  
  let ResourceCycles = [ 1];  
}  
def : WriteRes<WriteMADD, [Multiplier, Adder]> {  
  let ResourceCycles = [ 2, 3];  
}
```

ResourceSegments

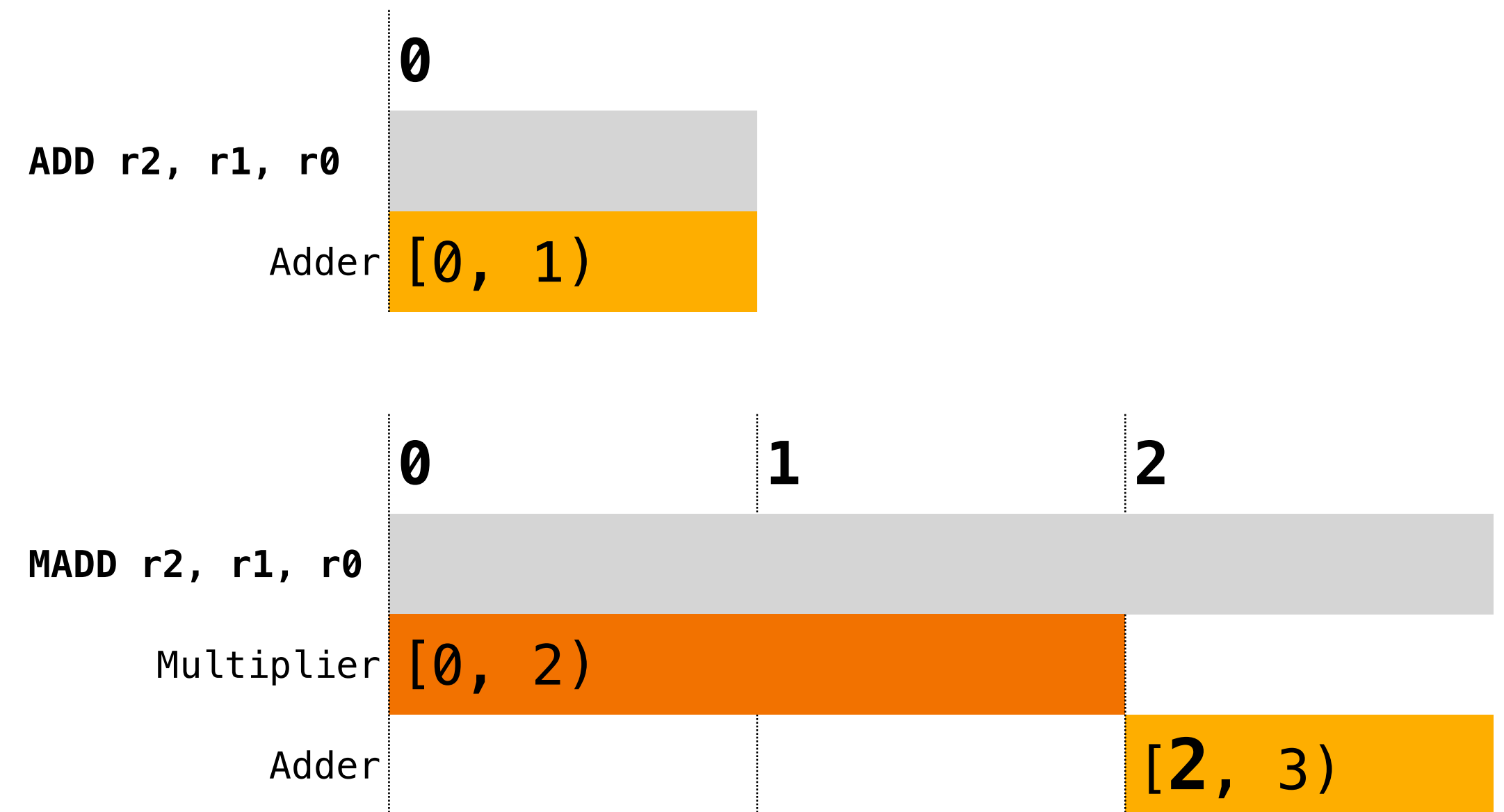
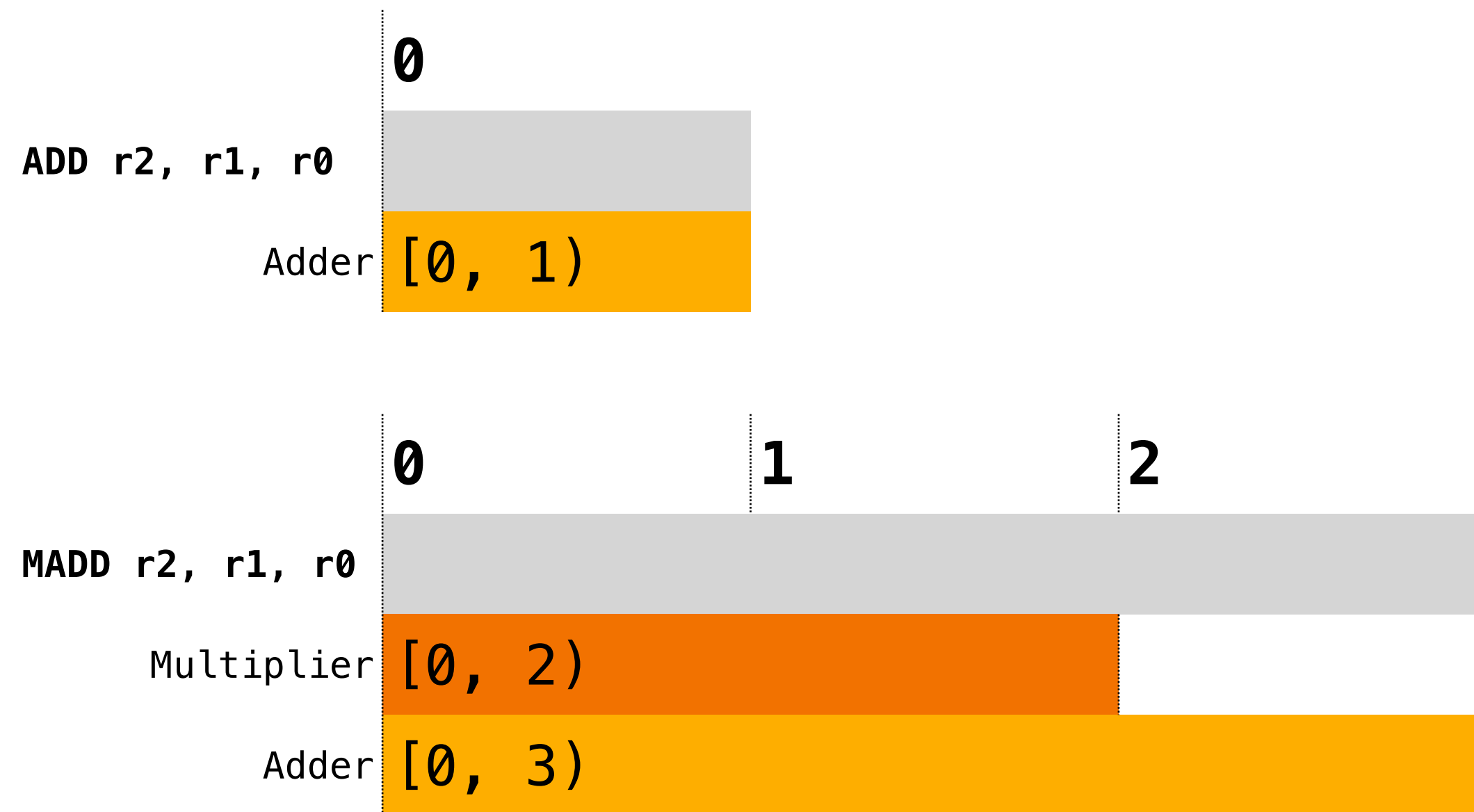
Thinking in terms of intervals, open on the right: [A, B)



```
def : WriteRes<WriteADD, [Adder]> {  
  let ResourceCycles = [ 1];  
}  
def : WriteRes<WriteMADD, [Multiplier, Adder]> {  
  let ResourceCycles = [ 2, 3];  
}
```

ResourceSegments

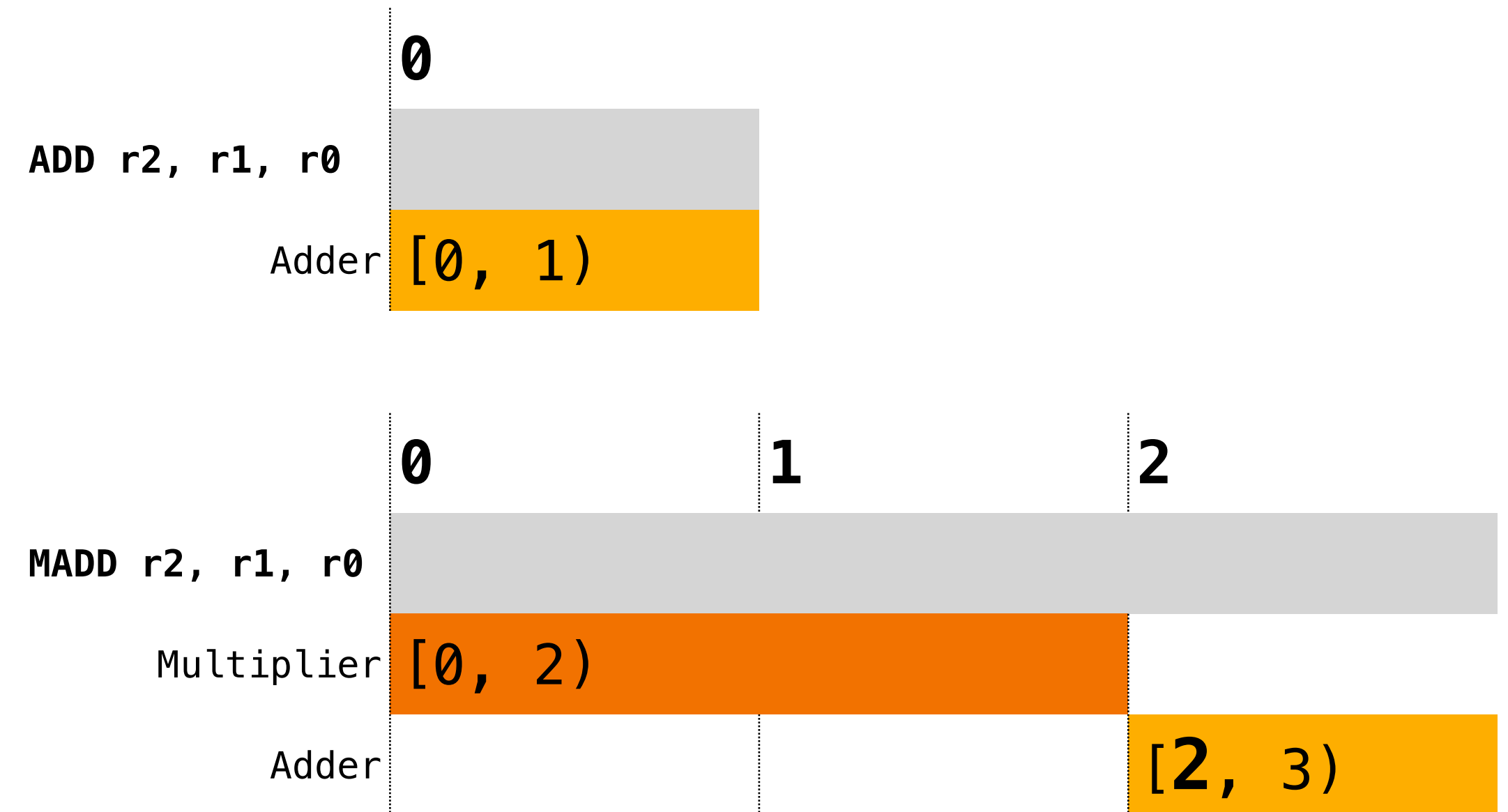
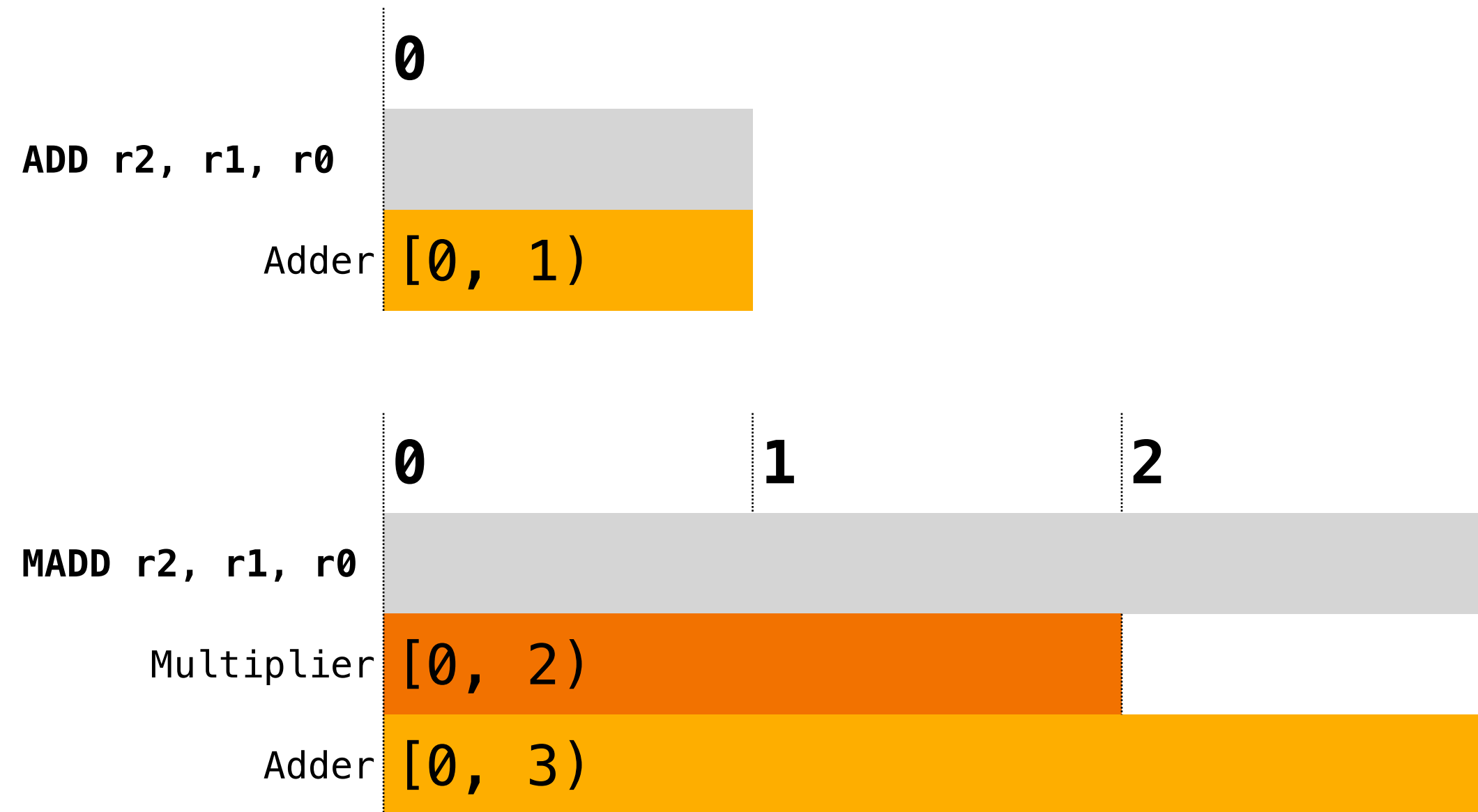
Thinking in terms of intervals, open on the right: [A, B)



```
def : WriteRes<WriteADD, [Adder]> {  
  let ResourceCycles = [ 1];  
}  
def : WriteRes<WriteMADD, [Multiplier, Adder]> {  
  let ResourceCycles = [ 2, 3];  
}
```


ResourceSegments

Thinking in terms of intervals, open on the right: [A, B)



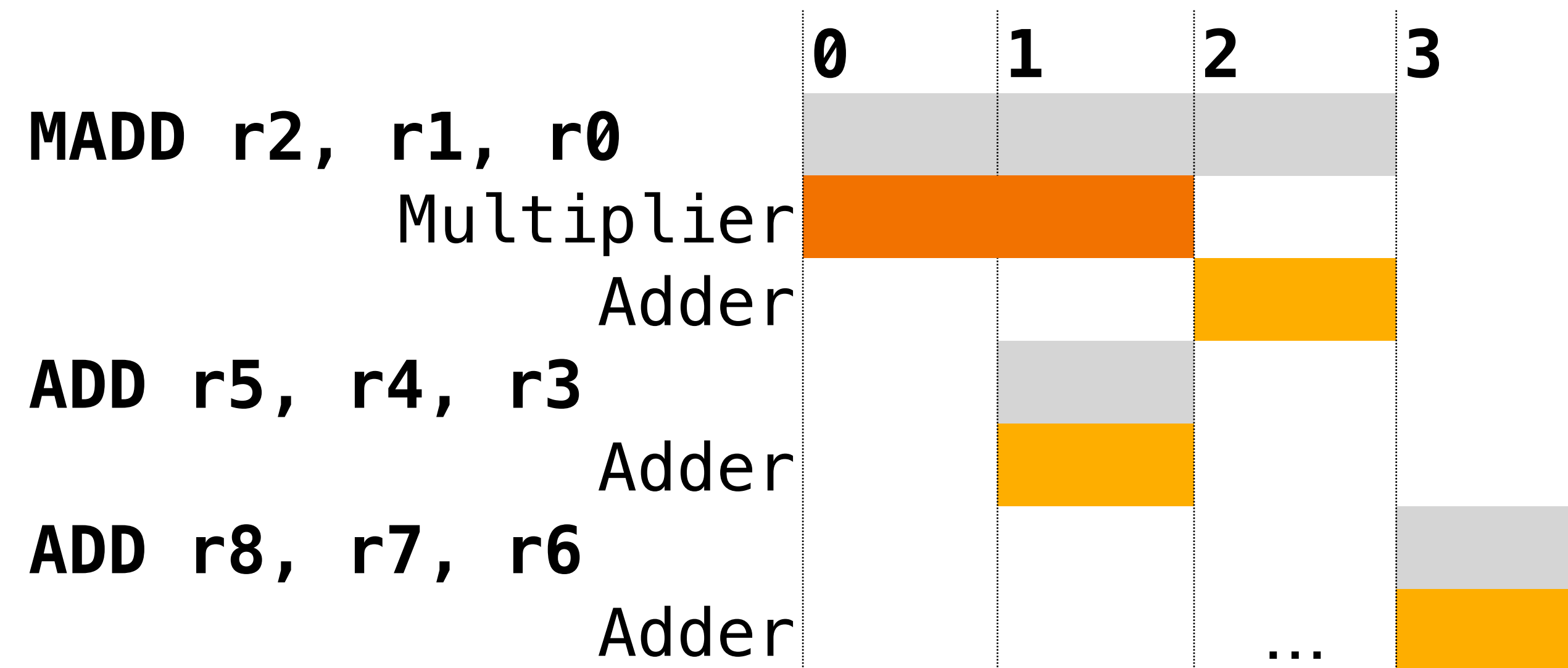
```
def : WriteRes<WriteADD, [Adder]> {
  let ResourceCycles = [ 1];
}
def : WriteRes<WriteMADD, [Multiplier, Adder]> {
  let ResourceCycles = [ 2, 3];
}
```

```
def : WriteRes<WriteADD, [Adder]> {
  let ResourceCycles = [1];
  let StartAtCycle = [0];
}
def : WriteRes<WriteMADD, [Multiplier, Adder]> {
  let ResourceCycles = [2, 3];
  let StartAtCycle = [0, 2];
}
```

Intermission

Advertise a new feature

From fancy tables...



...to text tables!

```
*** Final schedule for %bb.0 ***
* Schedule table (TopDown):
i: issue
x: resource booked
Cycle          |0      |1      |2      |3      |
MADD r2, r1, r0 | i     |       |       |       |
                Multiplier | x     | x     |       |       |
                Adder     |       |       | x     |       |
ADD r5, r4, r3  |       | i     |       |       |
                Adder     |       | x     |       |       |
ADD r8, r7, r6  |       |       |       | i     |
                Adder     |       |       |       | x     |
```

Debug messages generated by the compiler!

```
*** Final schedule for %bb.0 ***
* Schedule table (TopDown):
i: issue
x: resource booked
Cycle          |0      |1      |2      |3      |
MADD r2, r1, r0 | i     |       |       |       |
                Multiplier | x     | x     |       |       |
                Adder     |       |       | x     |       |
ADD r5, r4, r3  |       | i     |       |       |
                Adder     |       | x     |       |       |
ADD r8, r7, r6  |       |       |       | i     |
                Adder     |       |       |       | x     |
```

llc -misched-dump-schedule-trace

LIT unit tests for resource usage in scheduling models

```
# CHECK-LABEL: *** Final schedule for %bb.0 ***
# CHECK-NEXT: * Schedule table (TopDown):
# CHECK-NEXT:  i: issue
# CHECK-NEXT:  x: resource booked
# CHECK-NEXT: Cycle          |0    |1    |2    |3    |
# CHECK-NEXT: MADD r2, r1, r0   | i   |    |    |    |
# CHECK-NEXT:      Multiplier | x   | x   |    |    |
# CHECK-NEXT:      Adder      |    |    | x   |    |
# CHECK-NEXT: ADD r5, r4, r3      |    | i   |    |    |
# CHECK-NEXT:      Adder      |    | x   |    |    |
# CHECK-NEXT: ADD r8, r7, r6   |    |    |    | i   |
# CHECK-NEXT:      Adder      |    |    |    | x   |
```


Implementation

What changes in the code

TableGen representation and MachineScheduler

- TableGen:
 - `list<int> StartAtCycle = [];` added to the `WriteRes` class;
 - Backend changes in `llvm/utils/TableGen/SubtargetEmitter.cpp`

What changes in the code

TableGen representation and MachineScheduler

- TableGen:
 - `list<int> StartAtCycle = []`; added to the `WriteRes` class;
 - Backend changes in `llvm/utils/TableGen/SubtargetEmitter.cpp`
- MachineScheduler:
 - Data structure to handle intervals
 - New fine grain bookkeeping algorithm

Fine grain bookkeeping

Keeping track of resource intervals across the schedule

Current algorithm

```
ADD r2, r1, r0
ADD r5, r4, r3
MADD r9, r8, r7, r6
ADD r12, r11, r10
```

Multiplier
Adder

Last Seen

New algorithm

```
ADD r2, r1, r0
ADD r5, r4, r3
MADD r9, r8, r7, r6
ADD r12, r11, r10
```

Multiplier
Adder

All Seen

Legend

Assumed

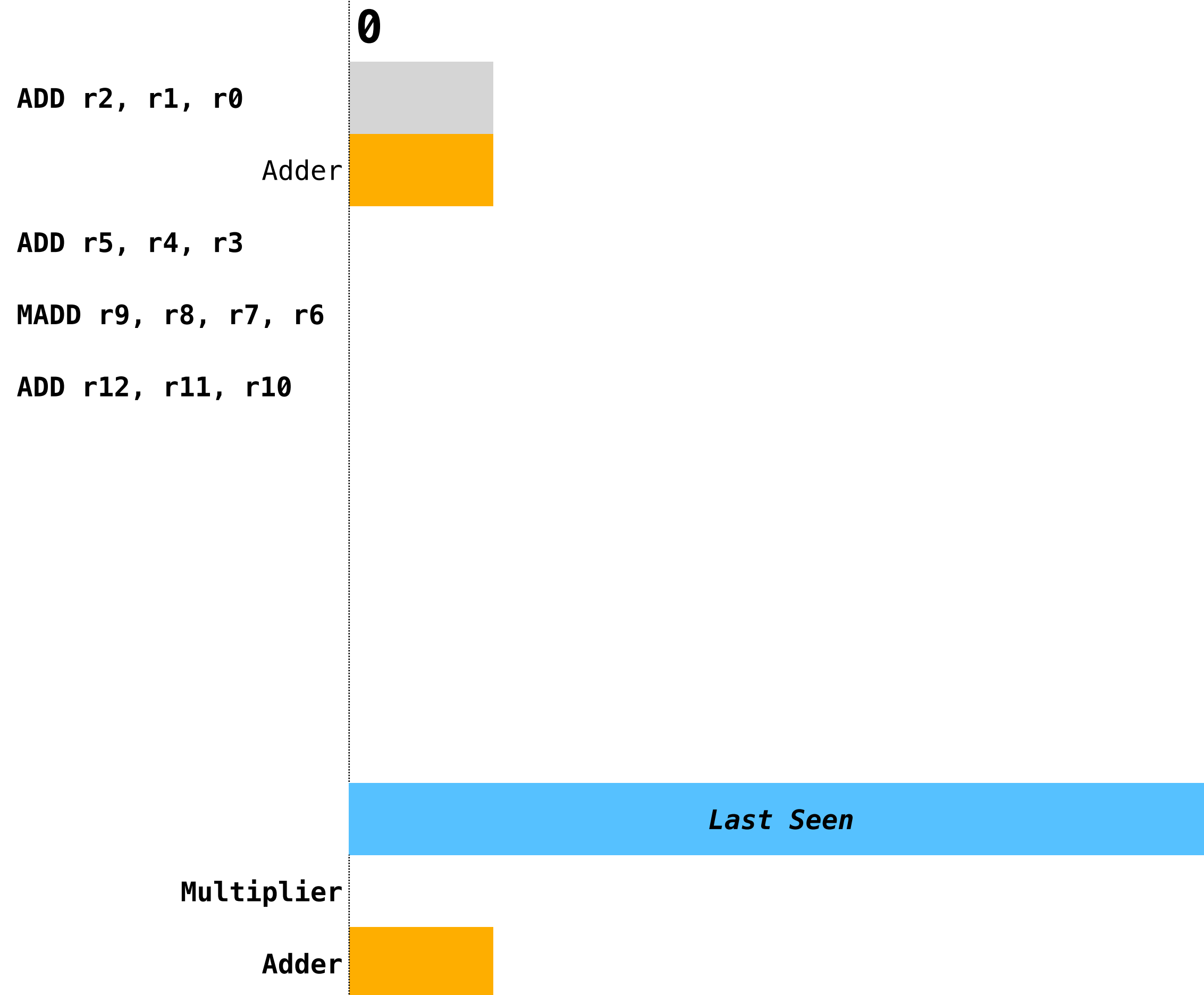
Seen

Multiplier

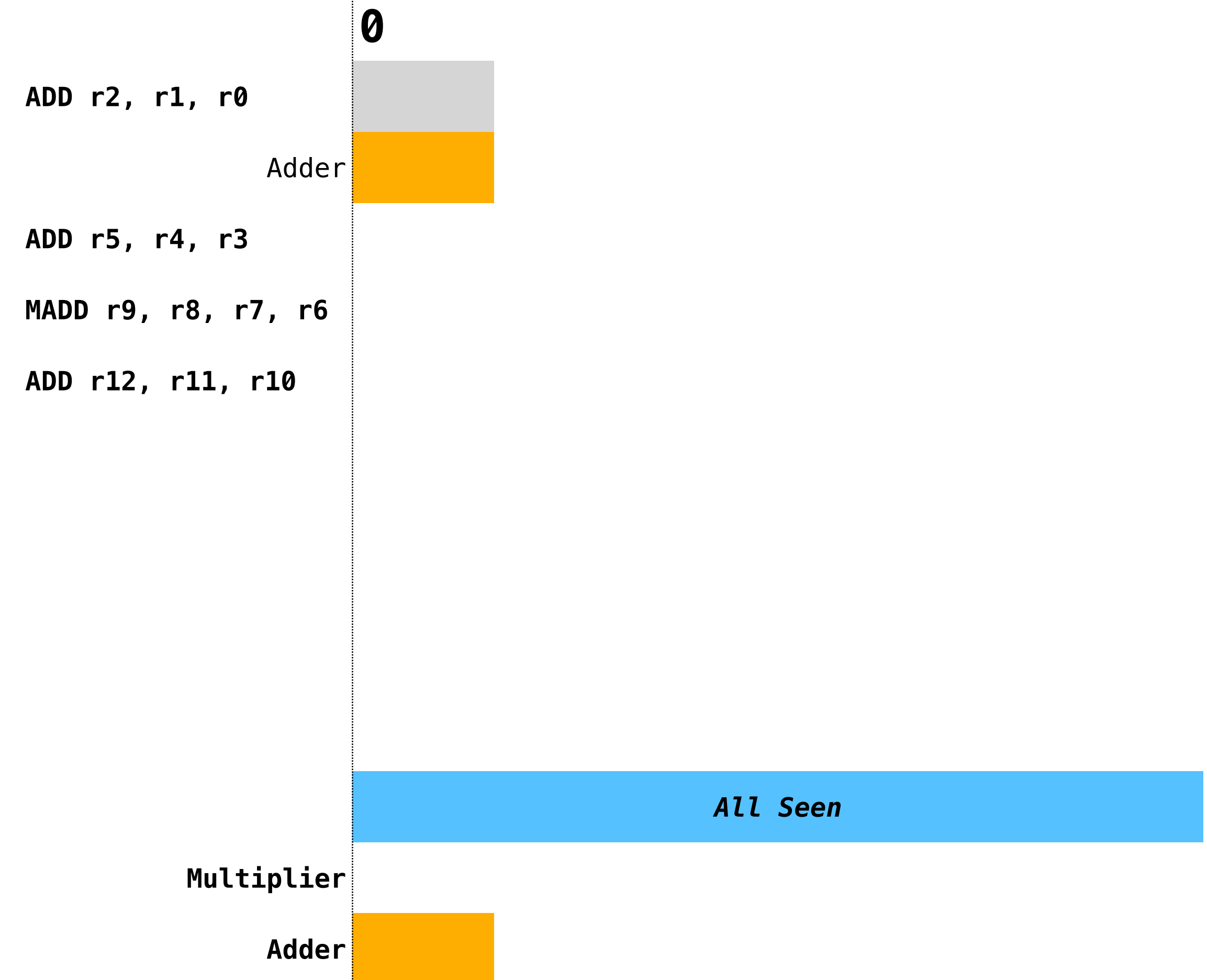
Adder



Current algorithm



New algorithm



Legend

Assumed

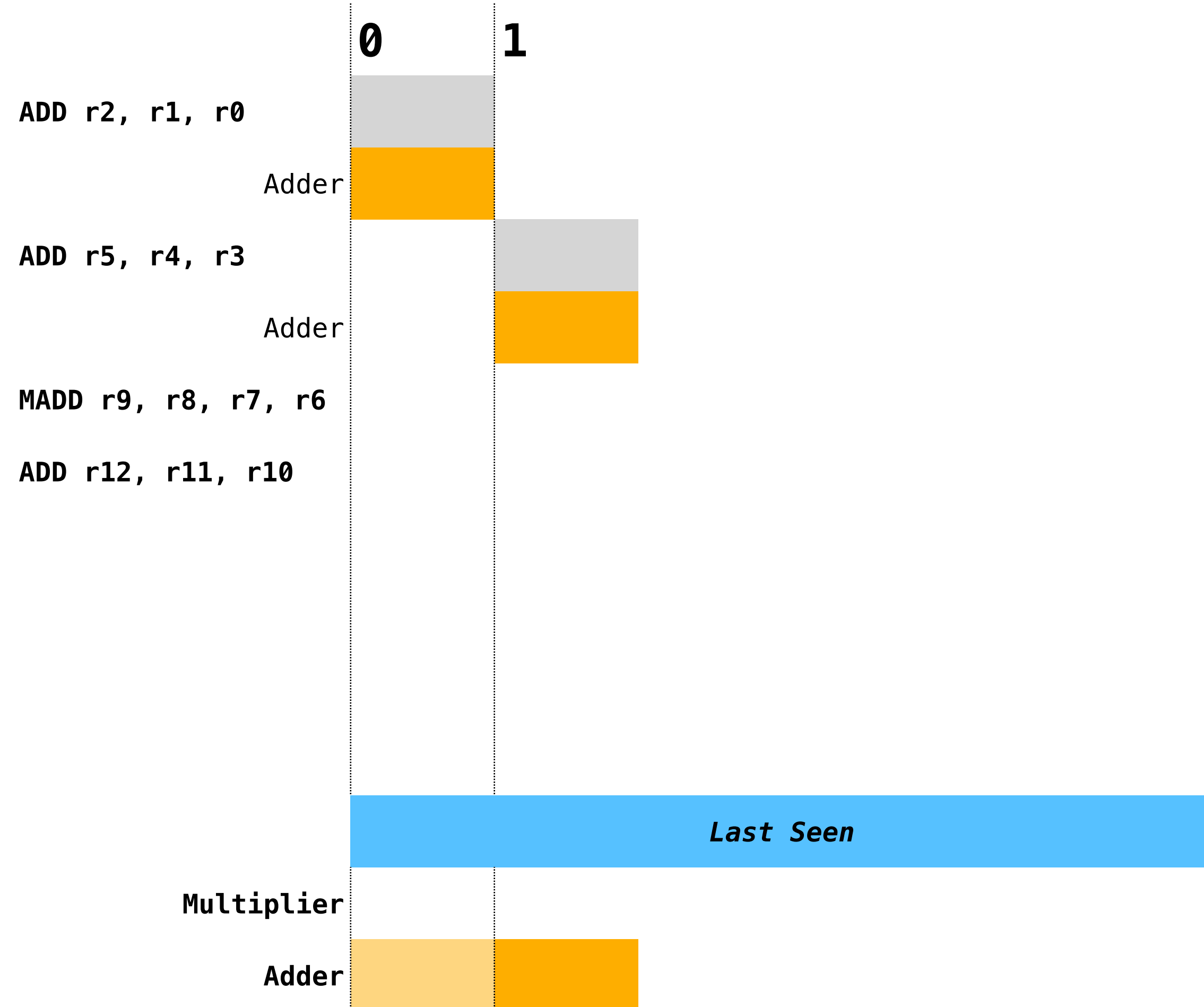
Seen

Multiplier

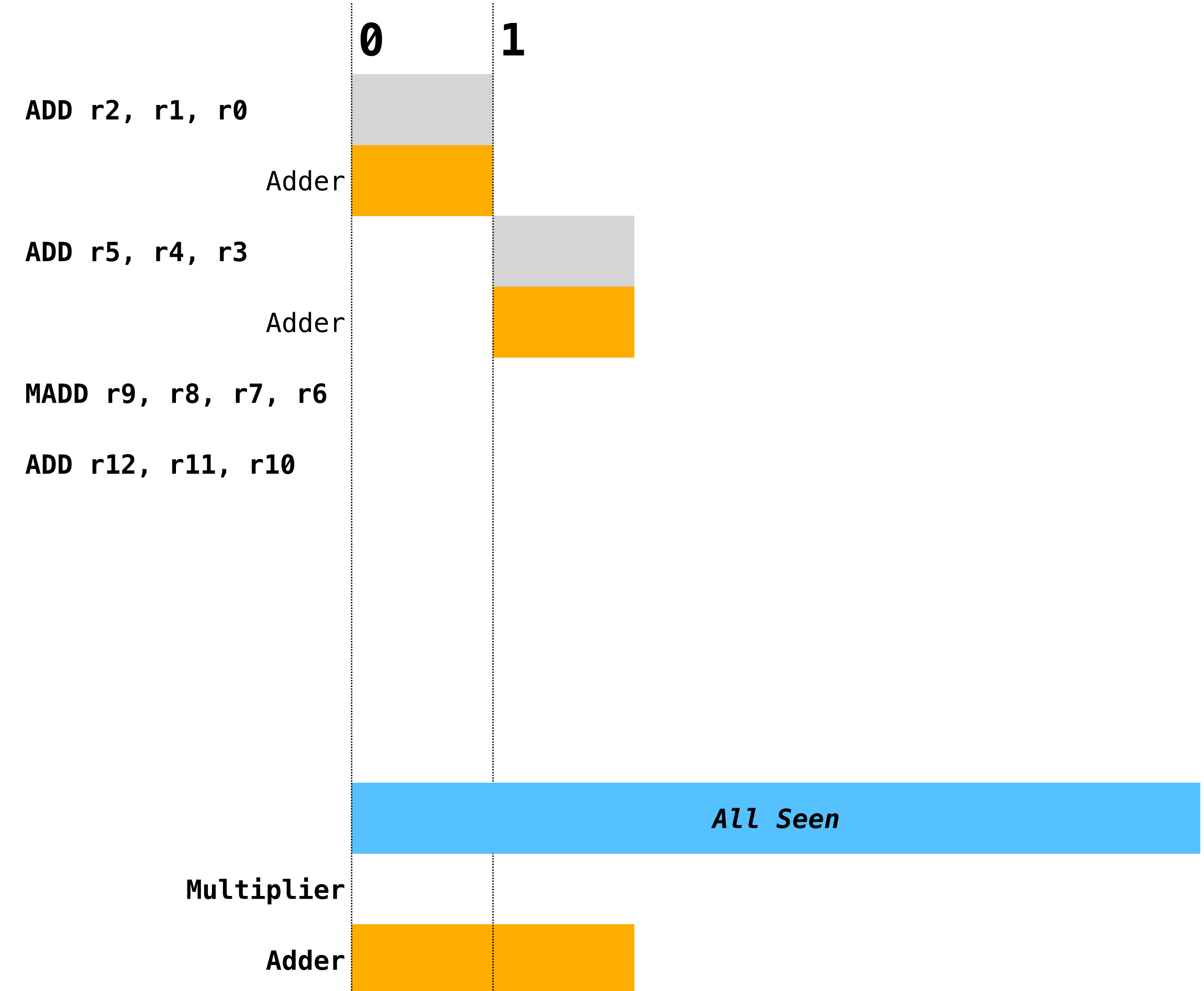
Adder



Current algorithm



New algorithm



Legend

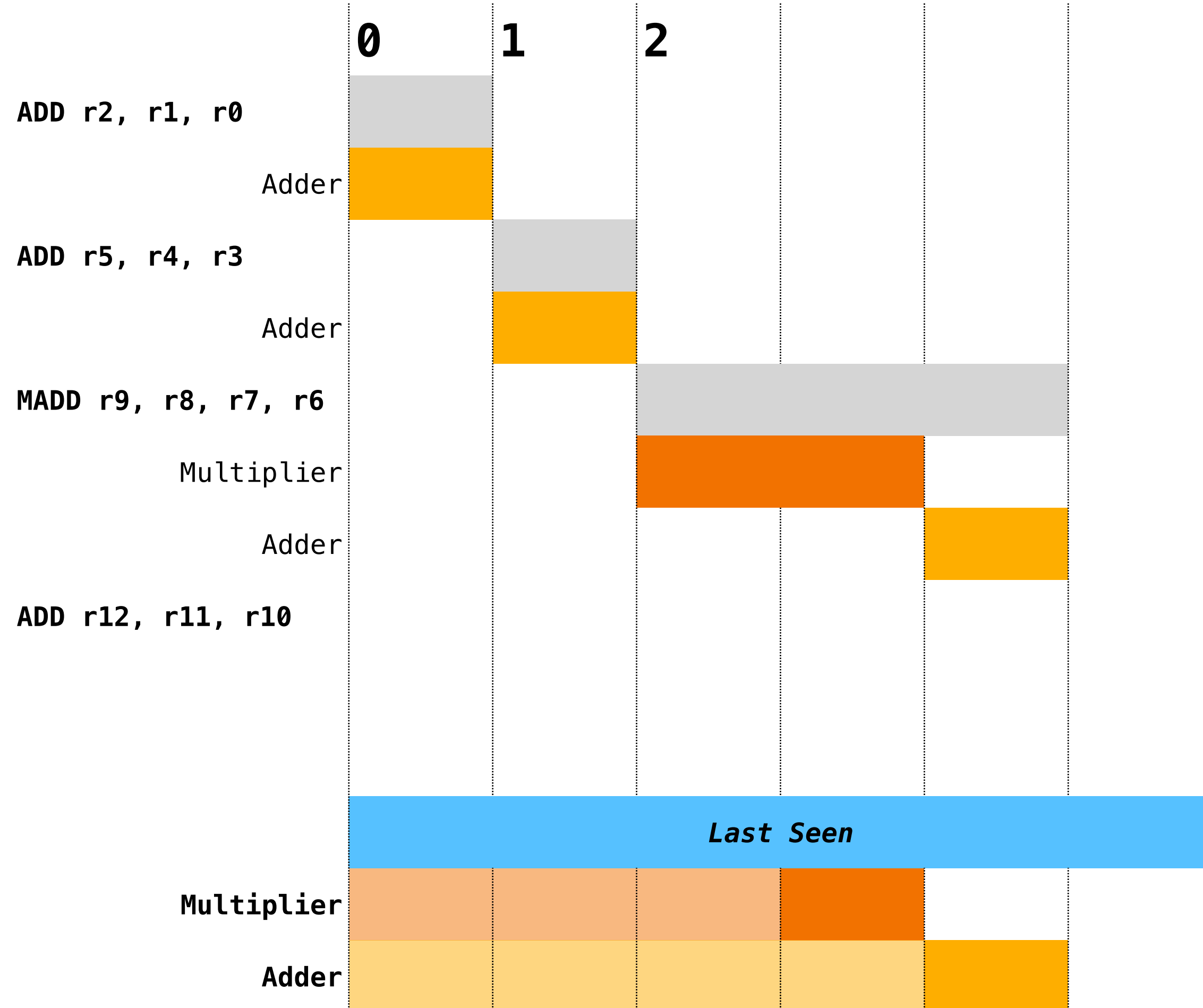
Assumed Seen

Multiplier

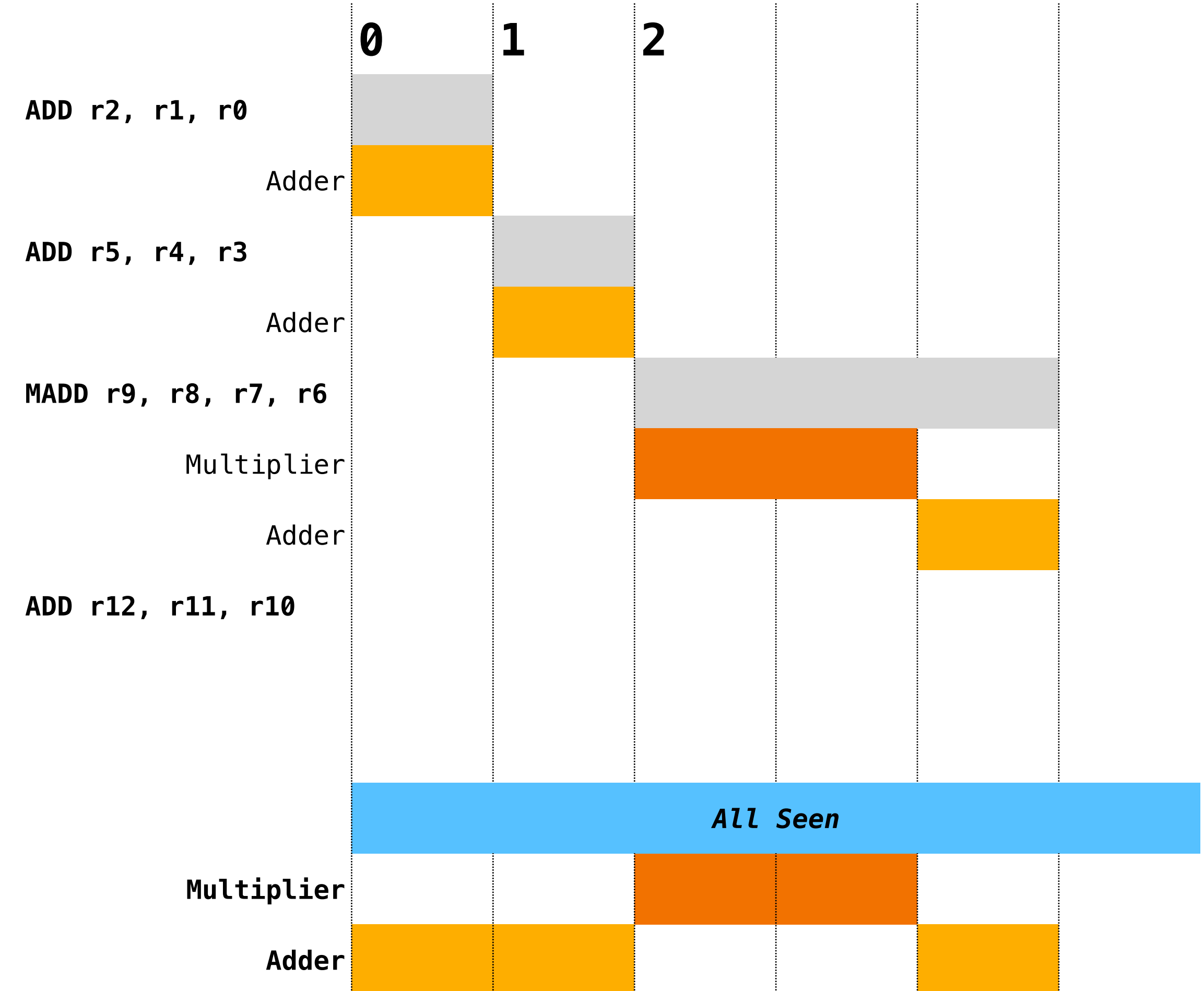
Adder



Current algorithm



New algorithm



Finds the gap in the disjoint interval!

Legend

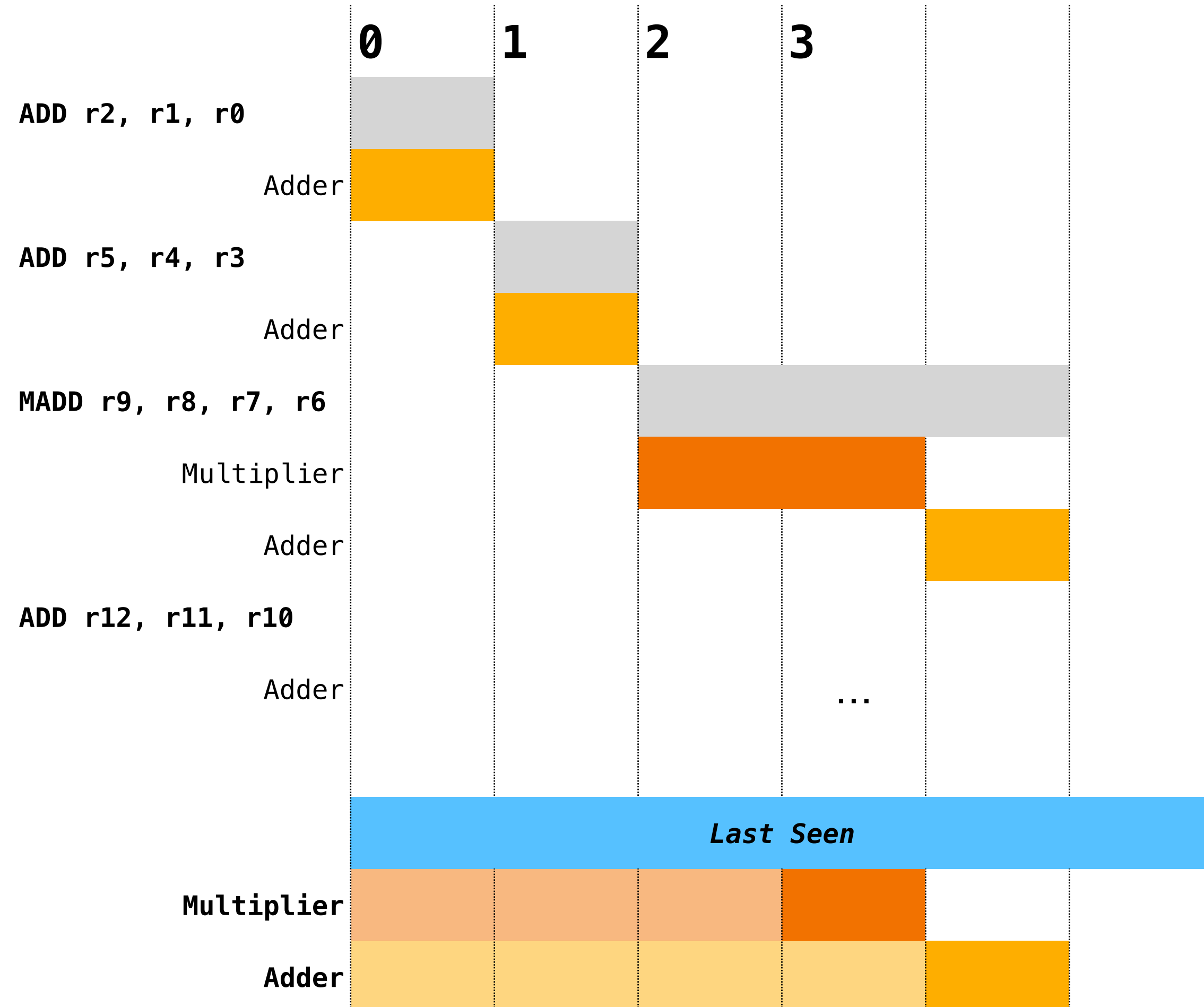
Assumed Seen

Multiplier

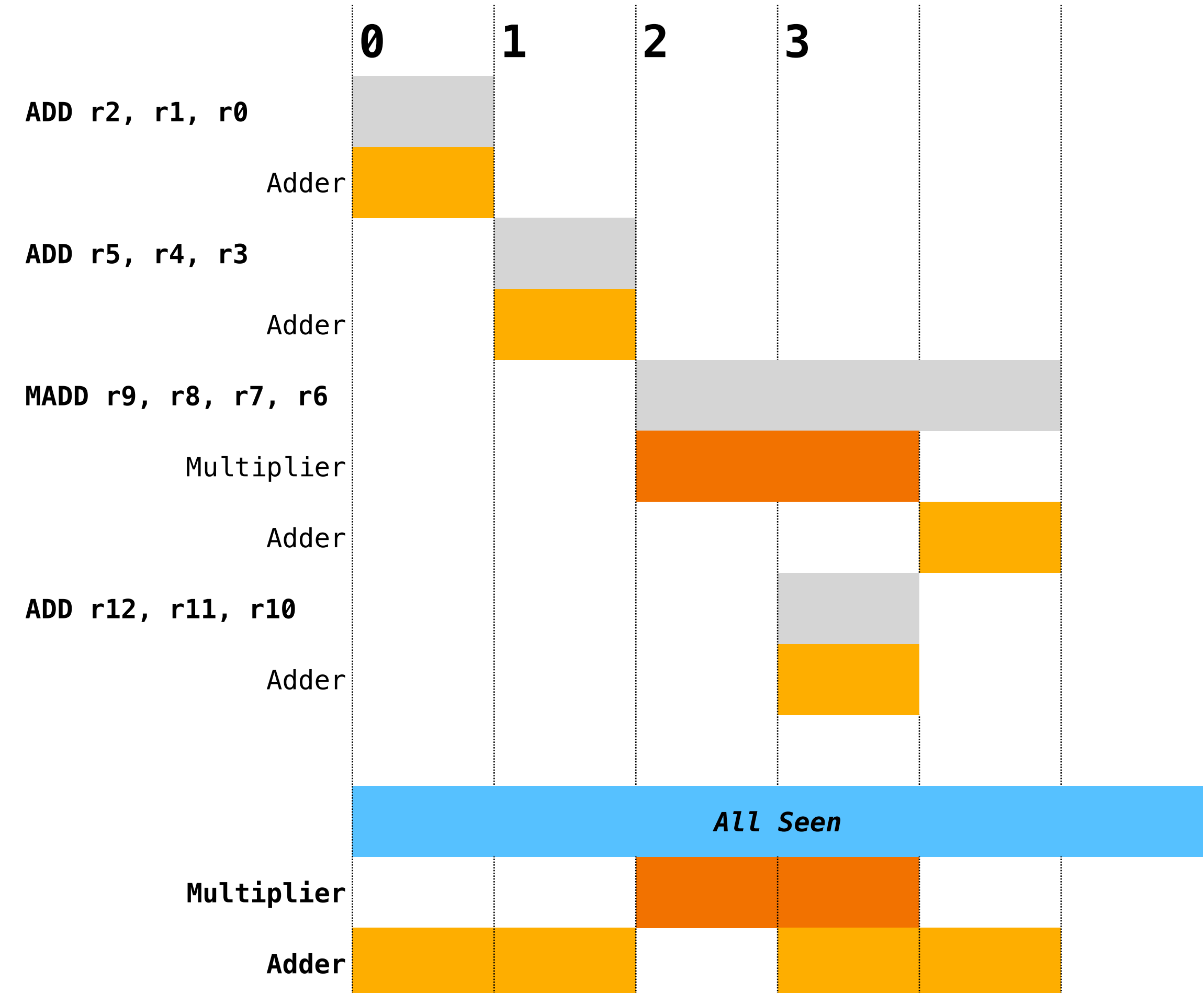
Adder



Current algorithm



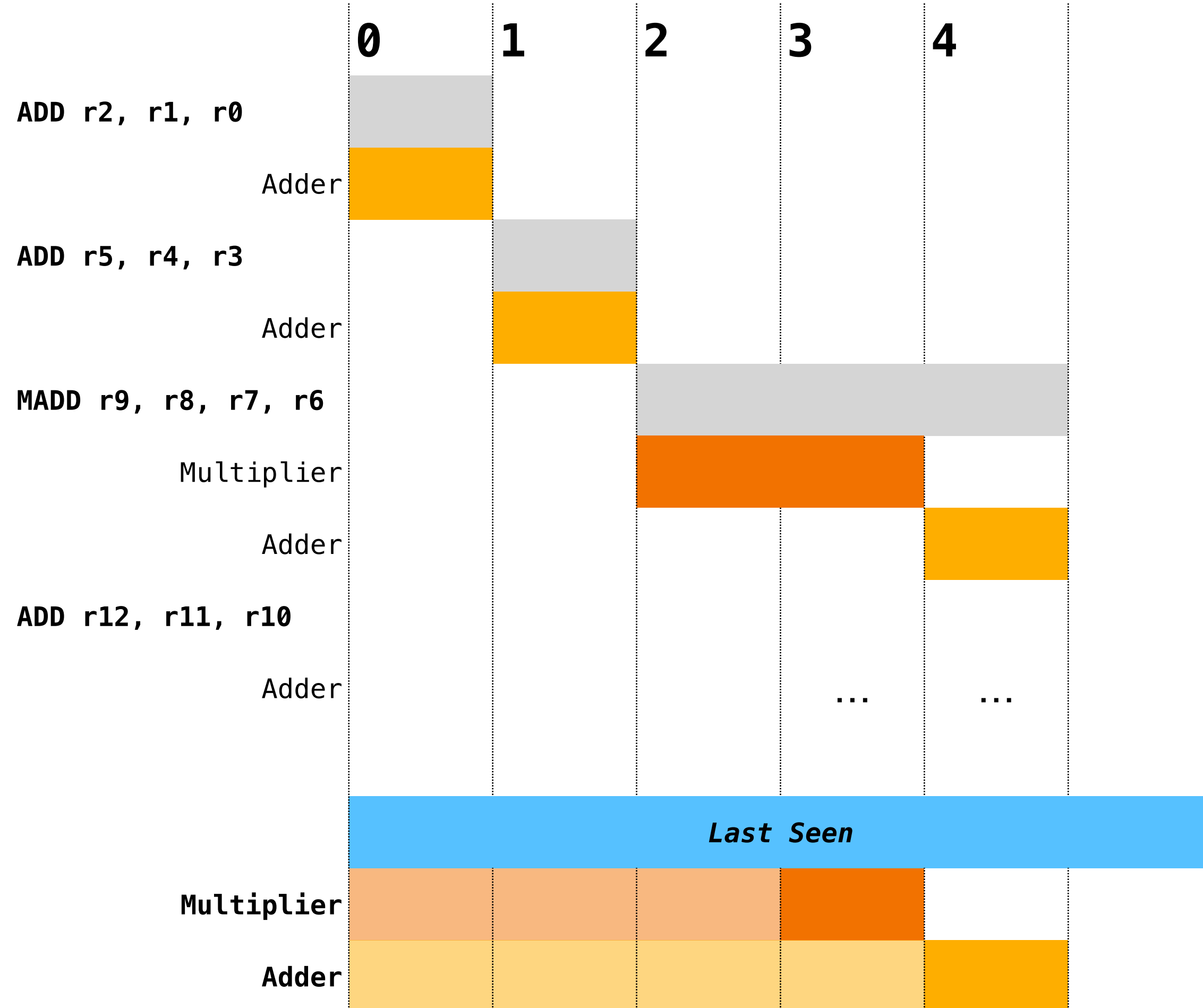
New algorithm



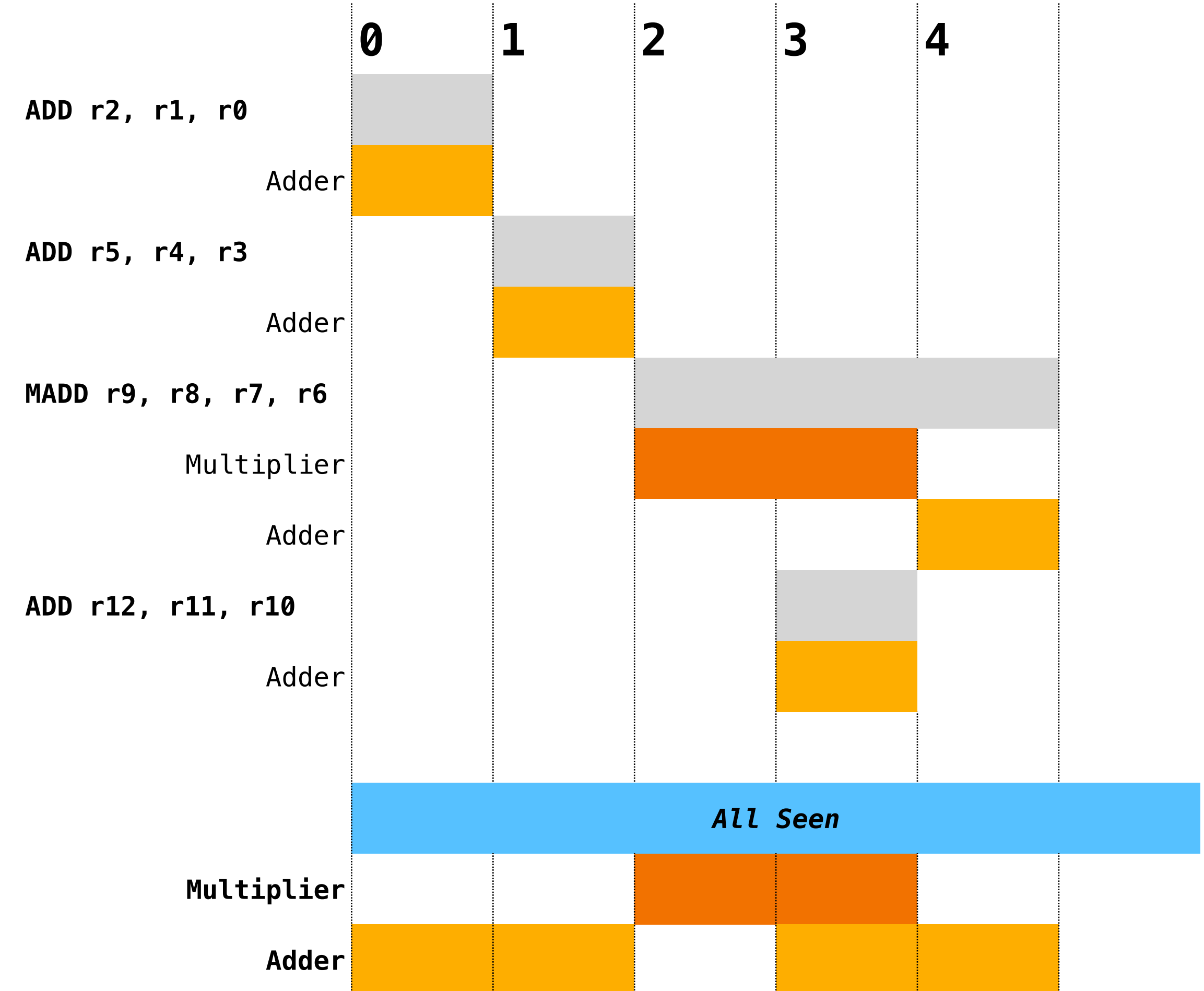
Legend

	Assumed	Seen
Multiplier		
Adder		

Current algorithm



New algorithm



Better estimate of execution.

Legend

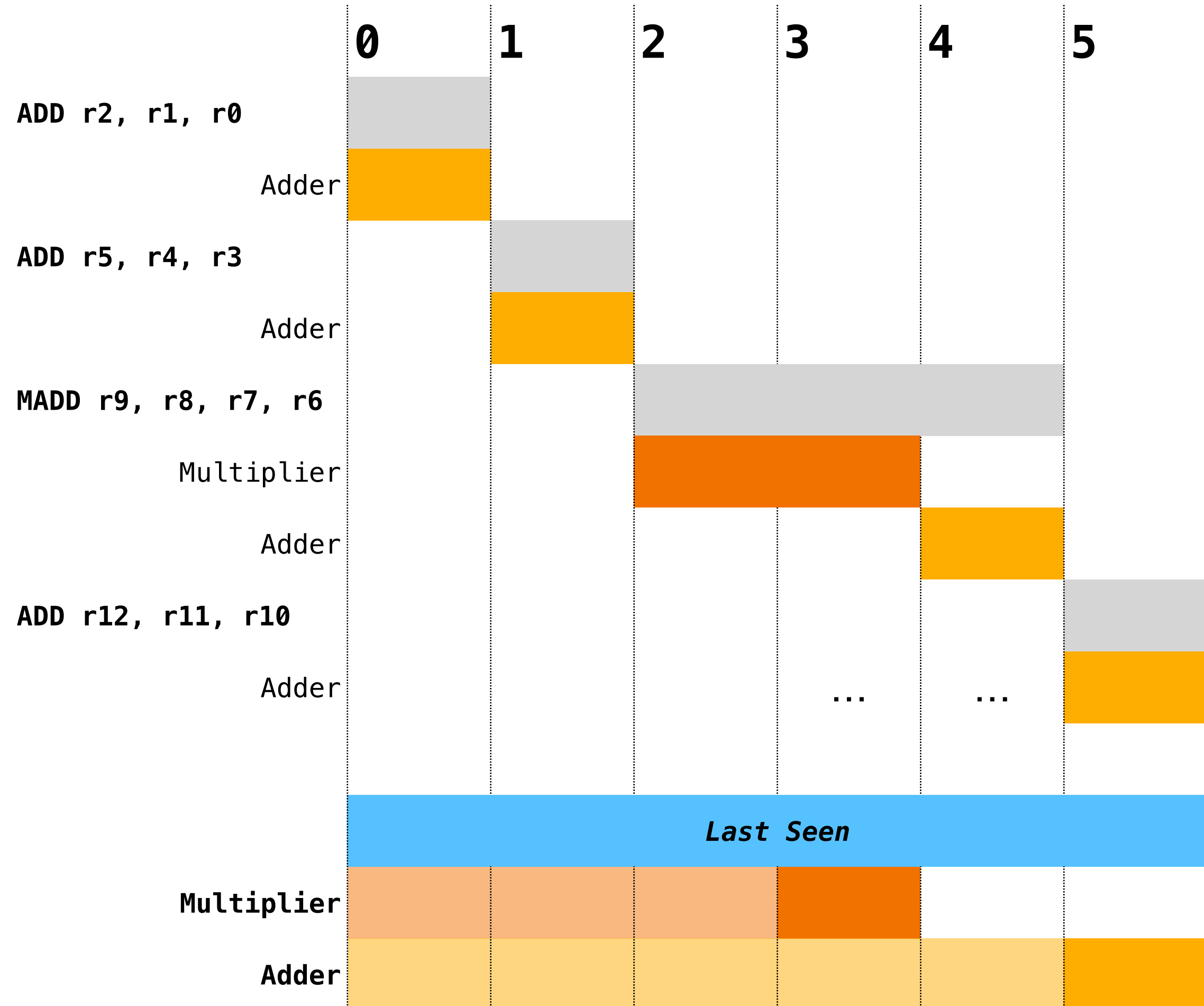
Assumed Seen

Multiplier

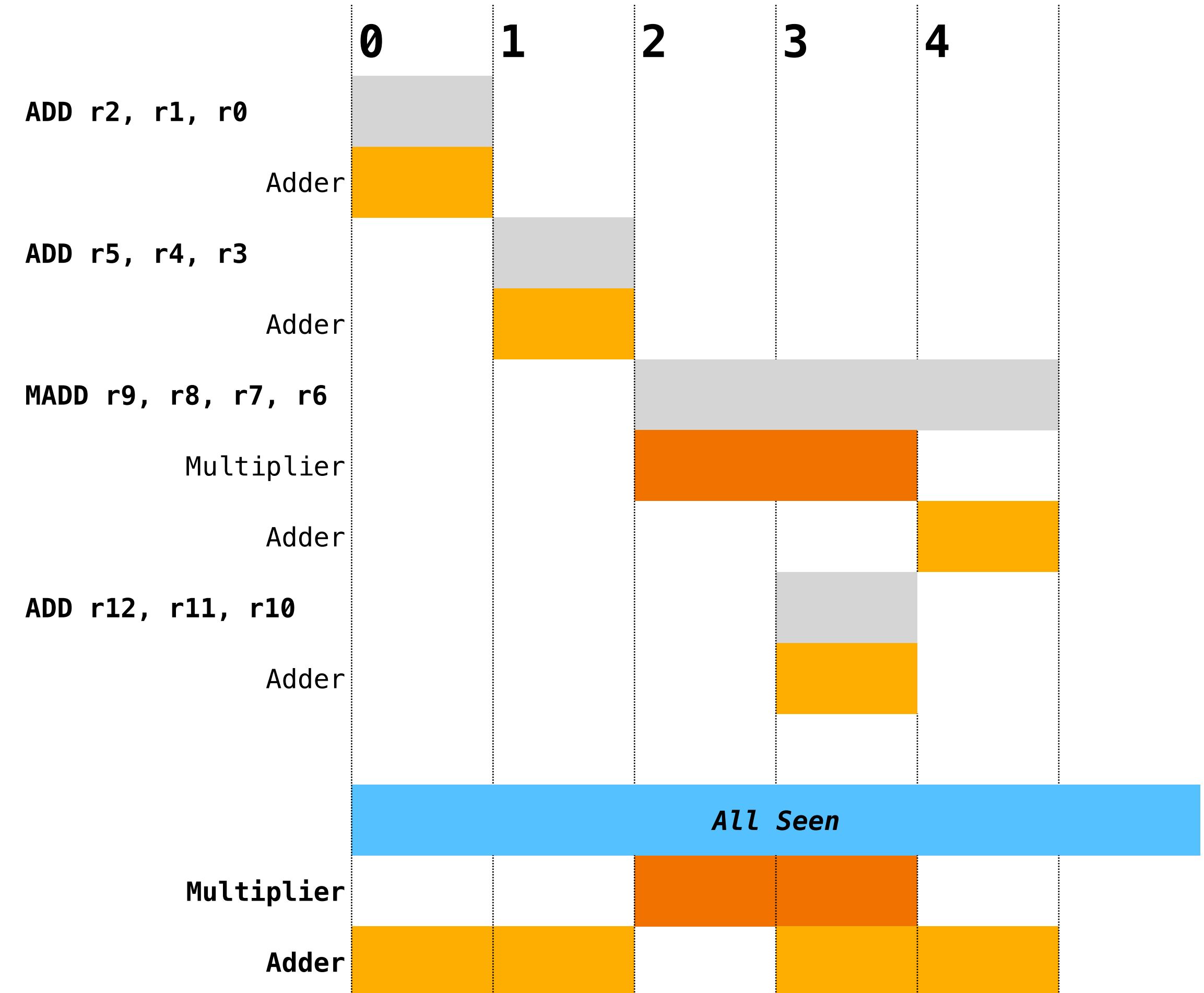
Adder



Current algorithm



New algorithm



Performance improvements

Example 1: from 25 cycles to 12 cycles

Top-down scheduling

bb.0:
 liveins: \$x9, \$x11, \$x13, \$x15, \$x17, \$x19
 \$x10 = ADD \$x9, \$x9
 \$x12 = SUB \$x11, \$x11
 \$x16 = SLL \$x15, \$x15
 \$x14 = MUL \$x13, \$x13
 \$x18 = SRL \$x17, \$x17

```
test001:bb.0
*** Final schedule for %bb.0 ***
* Schedule table (TopDown):
i: issue
x: resource booked
```

Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
ADD	i																								
ResX0	x																								
ResX1	x	x																							
ResX2	x	x	x																						
ResX3	x	x	x	x																					
ResX4	x	x	x	x	x																				
SUB						i																			
ResX2						x																			
ResX3						x	x																		
ResX4						x	x	x																	
ResX0						x	x	x	x																
ResX1						x	x	x	x	x															
SLL											i														
ResX1											x														
ResX2											x	x													
ResX3											x	x	x												
ResX4											x	x	x	x											
ResX0											x	x	x	x	x										
MUL																i									
ResX4											x	x	x			x									
ResX0											x	x	x			x									
ResX1											x	x	x			x									
ResX2											x	x	x			x									
ResX3											x	x	x			x									
SRL.																					i				
ResX3																x					x				
ResX4																x					x				
ResX0																x					x	x			
ResX1																x					x	x			
ResX2																x					x	x			

```
test001:bb.0
*** Final schedule for %bb.0 ***
* Schedule table (TopDown):
i: issue
x: resource booked
```

Cycle	0	1	2	3	4	5	6	7	8	9	10	11
ADD	i											
ResX0	x											
ResX1		x										
ResX2			x									
ResX3				x								
ResX4					x							
SUB		i										
ResX2		x										
ResX3			x									
ResX4				x								
ResX0					x							
ResX1						x						
SLL			i									
ResX1			x									
ResX2				x								
ResX3					x							
ResX4						x						
ResX0							x					
SRL.						i						
ResX3						x						
ResX4						x						
ResX0							x					
ResX1								x				
ResX2									x			
MUL								i				
ResX4								x				
ResX0									x			
ResX1										x		
ResX2											x	
ResX3												x

Example 2: from 17 cycles to 7 cycles

Bottom-up scheduling

bb.0:

```
liveins: $x9, $x11, $x13, $x15, $x17, $x19
$x10 = ADD $x9, $x9
$x12 = SUB $x11, $x11
$x14 = MUL $x13, $x13
$x16 = SLL $x15, $x15
$x18 = SRL $x17, $x17
$x20 = DIV $x19, $x19
```

```
test001:%bb.0
*** Final schedule for %bb.0 ***
* Schedule table (BottomUp):
i: issue
x: resource booked
```

Cycle	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-1
ADD	i																
ResX0	x																
ResX1	x	x	x	x	x												
ResX3	x	x	x	x	x	x											
ResX2	x	x	x	x	x	x	x										
MUL								i									
ResX3								x									
ResX0								x	x								
ResX2								x	x	x							
SLL											i						
ResX2											x						
ResX0											x	x					
SUB													i				
ResX2													x				
ResX0													x				
SRL														i			
ResX3														x			
ResX2														x			
DIV.															i		
ResX1															x		
ResX0															x	x	

```
test001:%bb.0
*** Final schedule for %bb.0 ***
* Schedule table (BottomUp):
i: issue
x: resource booked
```

Cycle	5	4	3	2	1	0	-1
ADD	i						
ResX0	x						
ResX1		x	x	x	x		
ResX3						x	
ResX2							x
SUB		i					
ResX2		x					
ResX0			x				
MUL			i				
ResX3			x				
ResX0				x			
ResX2					x		
SLL				i			
ResX2				x			
ResX0					x	x	
SRL					i		
ResX3					x		
ResX2						x	
DIV						i	
ResX1						x	
ResX0						x	x

Average improvements

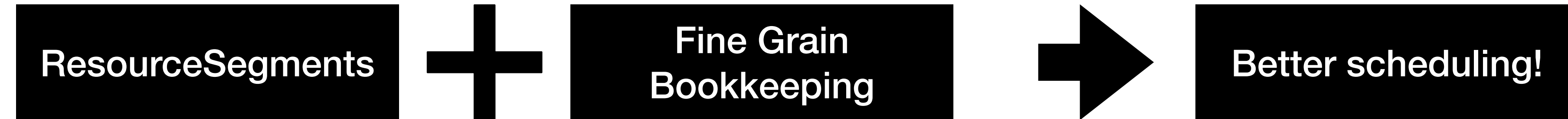
Artificial test cases (LIT)

TEST	TOP-DOWN (cycles)		BOTTOM-UP (cycles)	
	Current	New	Current	New
test-001	4	4	4	4
test-002	9	5	9	5
test-003	7	4	7	4
test-004	7	4	7	4
test-005	9	6	9	6
test-006	16	7	17	7
test-007.A	25	12	25	12
test-007.B	25	9	25	9
test-008	12	8	12	6
test-009	9	5	9	5
test-010	11	8	13	11
test-012	N/A	N/A	12	8
TOTAL	134	72	149	81
	New/Current = 0.53731		New/Current = 0.54362	

Recap

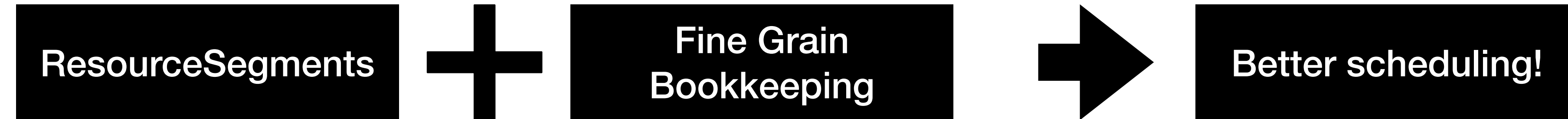
Better scheduling

...and testing!



Better scheduling

...and testing!



llc -misched-dump-schedule-trace

What's next

Adoption steps

- All current models are defaulted with `StartAtCycle = [0, ..., 0]`;
- Aim at replacing the current bookkeeping in the machine scheduler with the new one.
- Bit switch in the schedule model class to enable the new codepath.
- Further investigations:
 - Few CodeGen issues (it seems to find gaps that couldn't be found before)
 - Compile time (threshold of 10 intervals per resource).
- Work is ongoing, but WIP patches are up for review / feedback / try out

Reviews on Phabricator

Feedback is welcome!

- D150310: Adding StartAtCycle to WriteRes (NFC)
- D150311: Schedule traces in debug
- D150312: Modify MachineScheduler to use StartAtCycle

Thank you!

Questions?