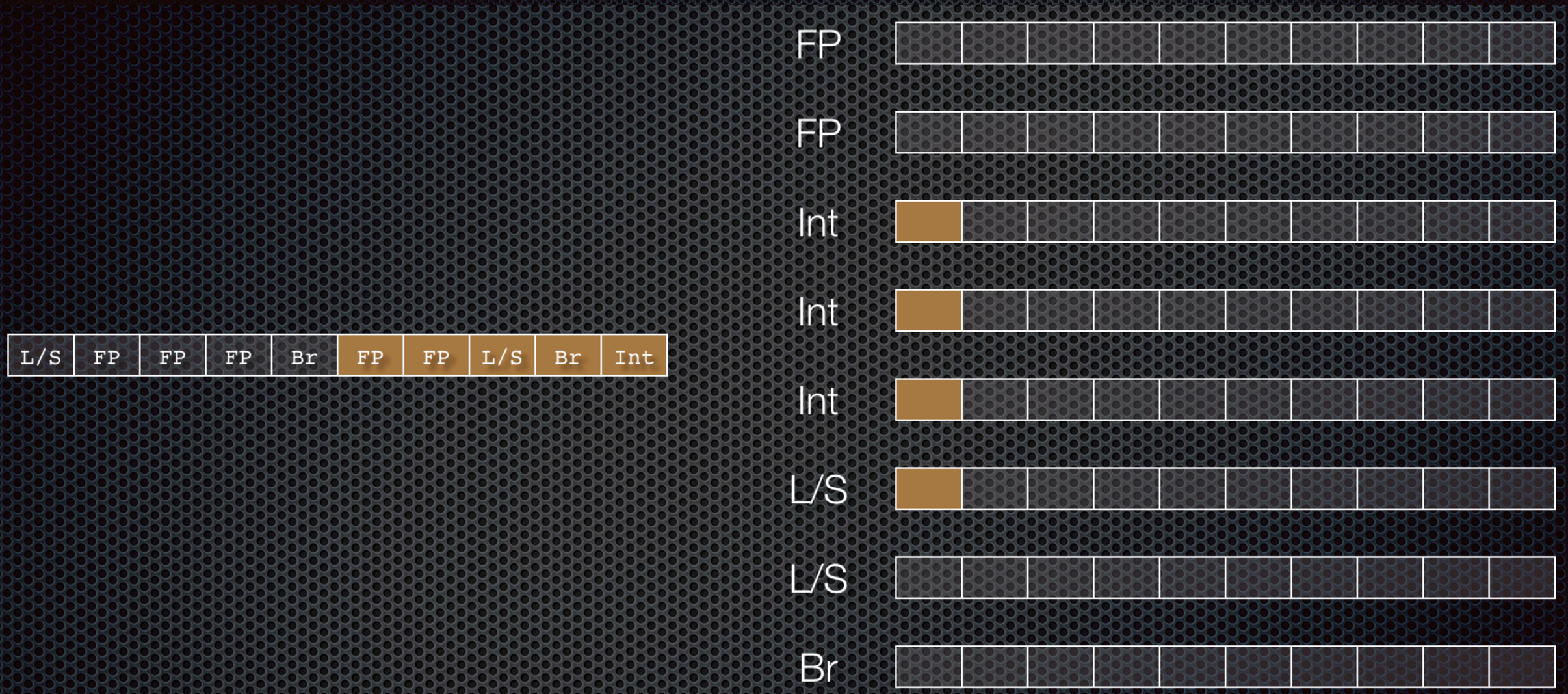
Fine-Grained Multithreading

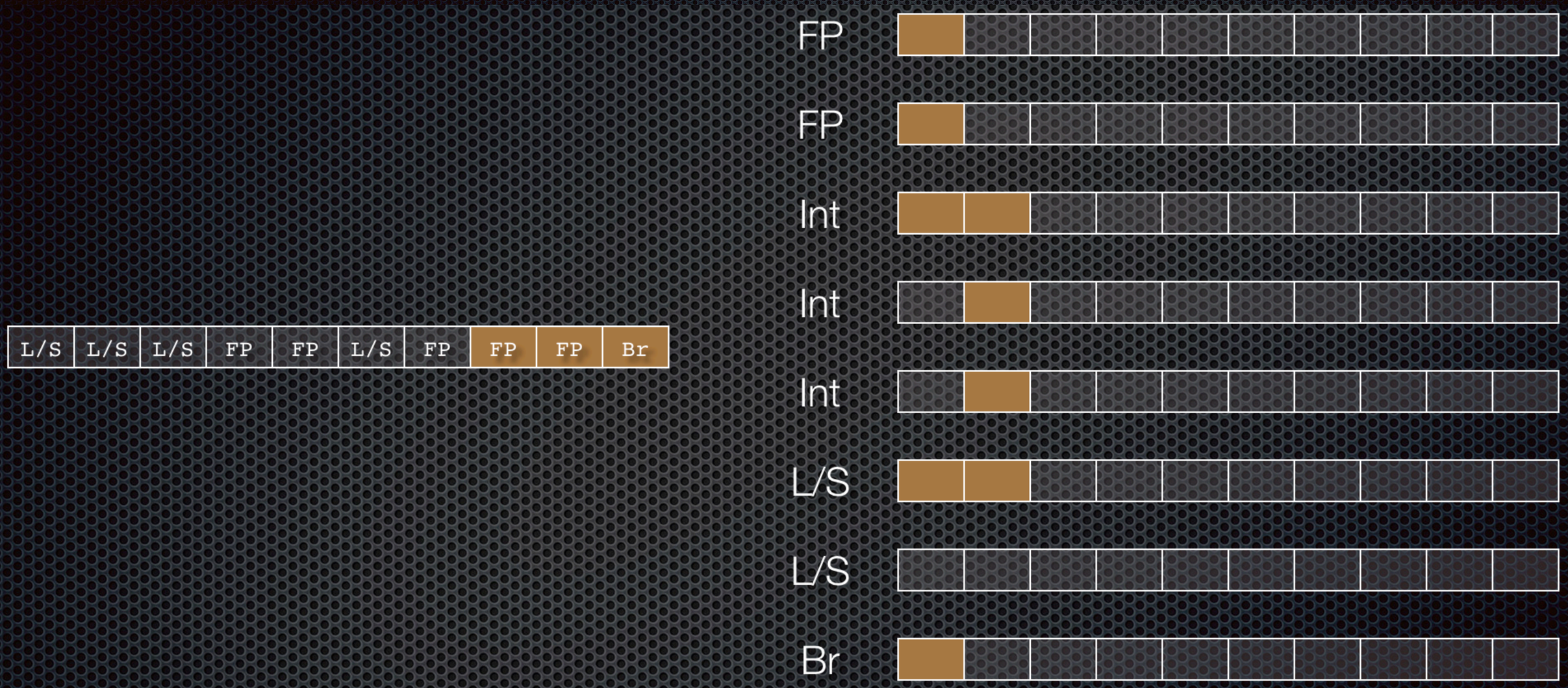
Filling Pipes with Instructions from Alternate Sources

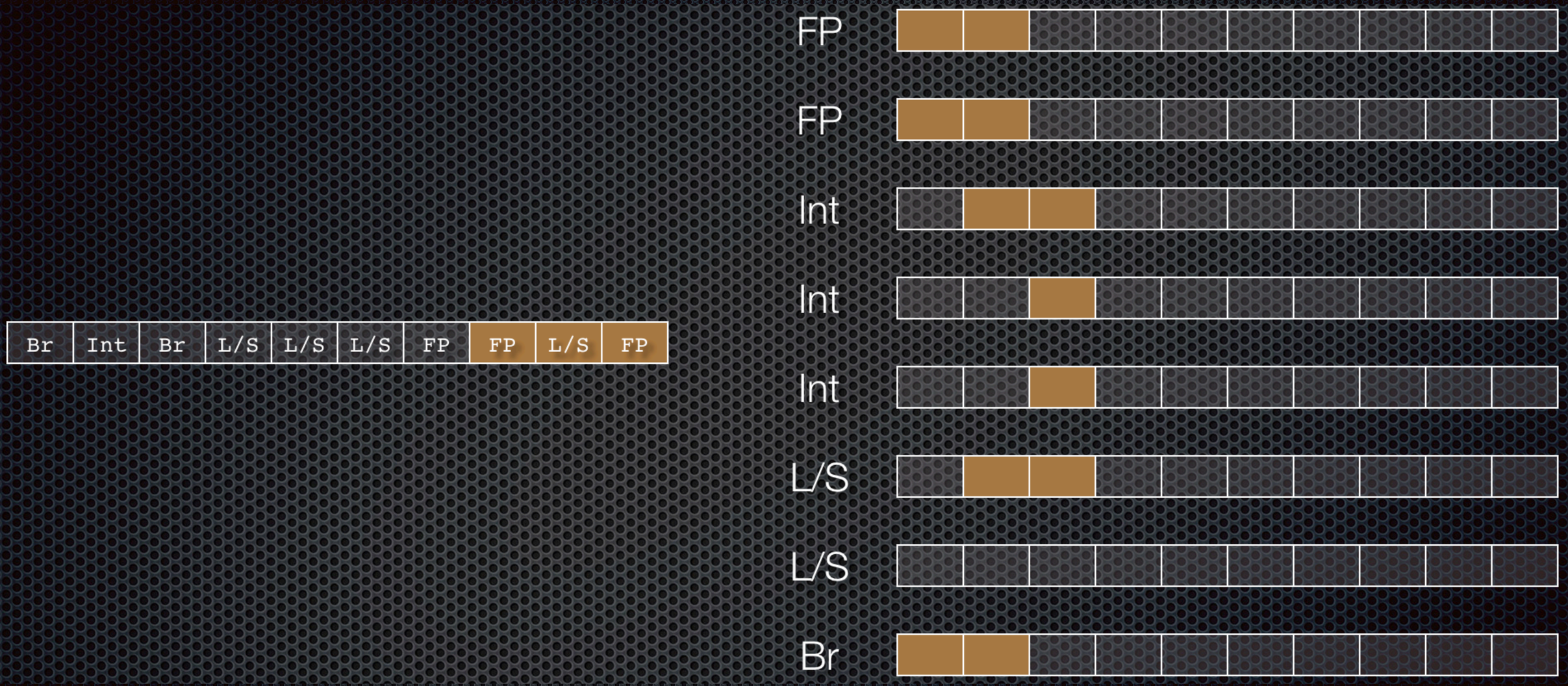
Superscalar Issue Example

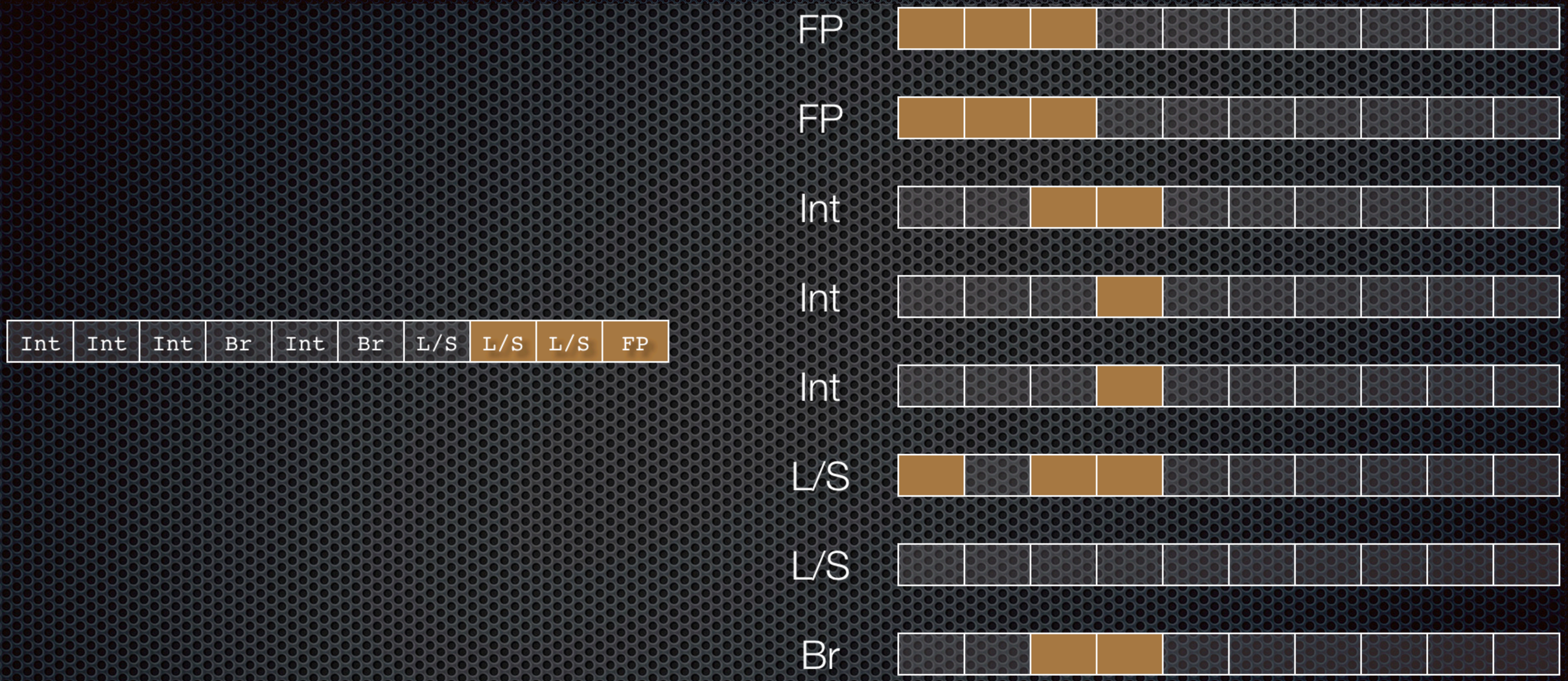
- Multiple pipelines with different functions
- Issue instructions as available from prefetch
- 8 pipes of depth 10
- To keep example simple, assume no stalls

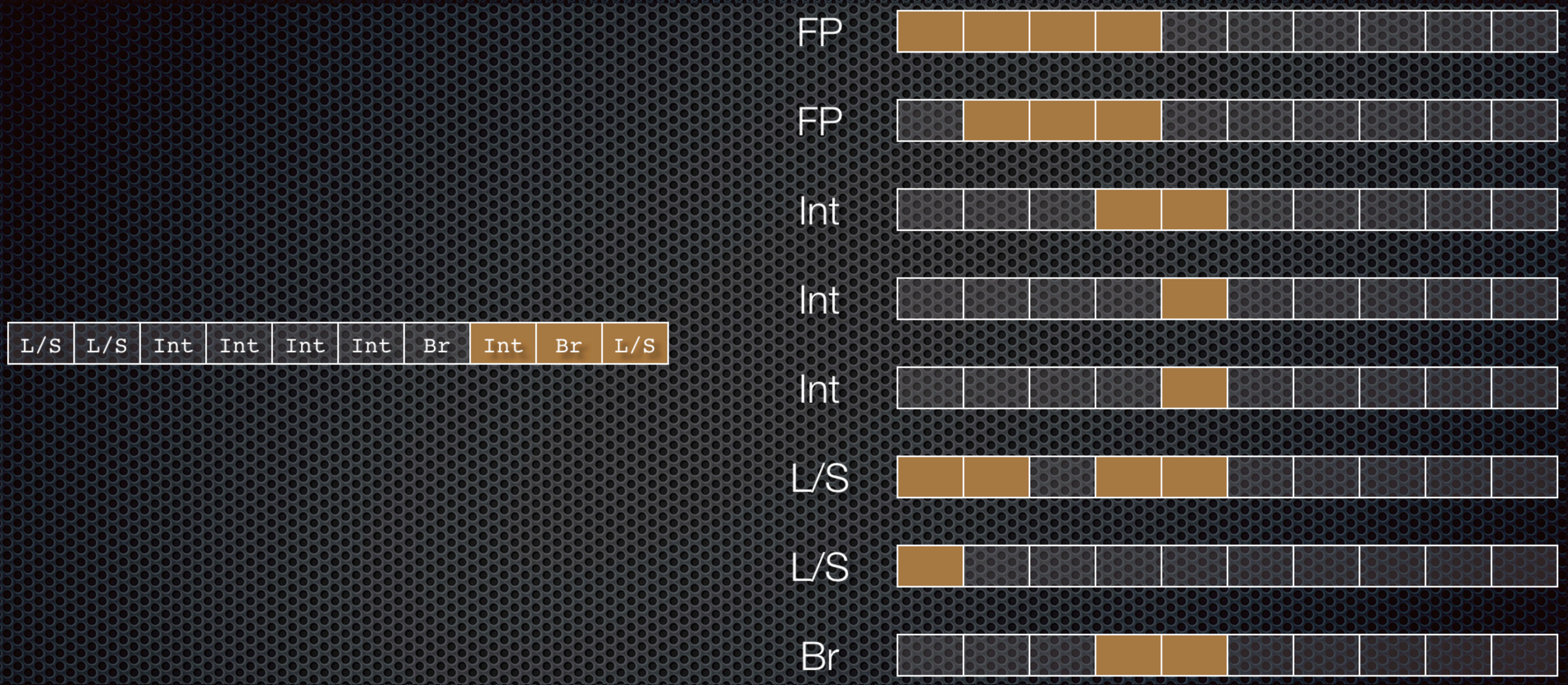

Br
FP
FP
L/S
Br
Int
L/S
Int
Int
Int
FP
FP
Int
Int
Int
L/S
L/S
Br


L/S
FP
FP
FP
Br
FP
FP
L/S
Br
Int
FP
FP
Int
Int
Int
L/S
L/S
Br

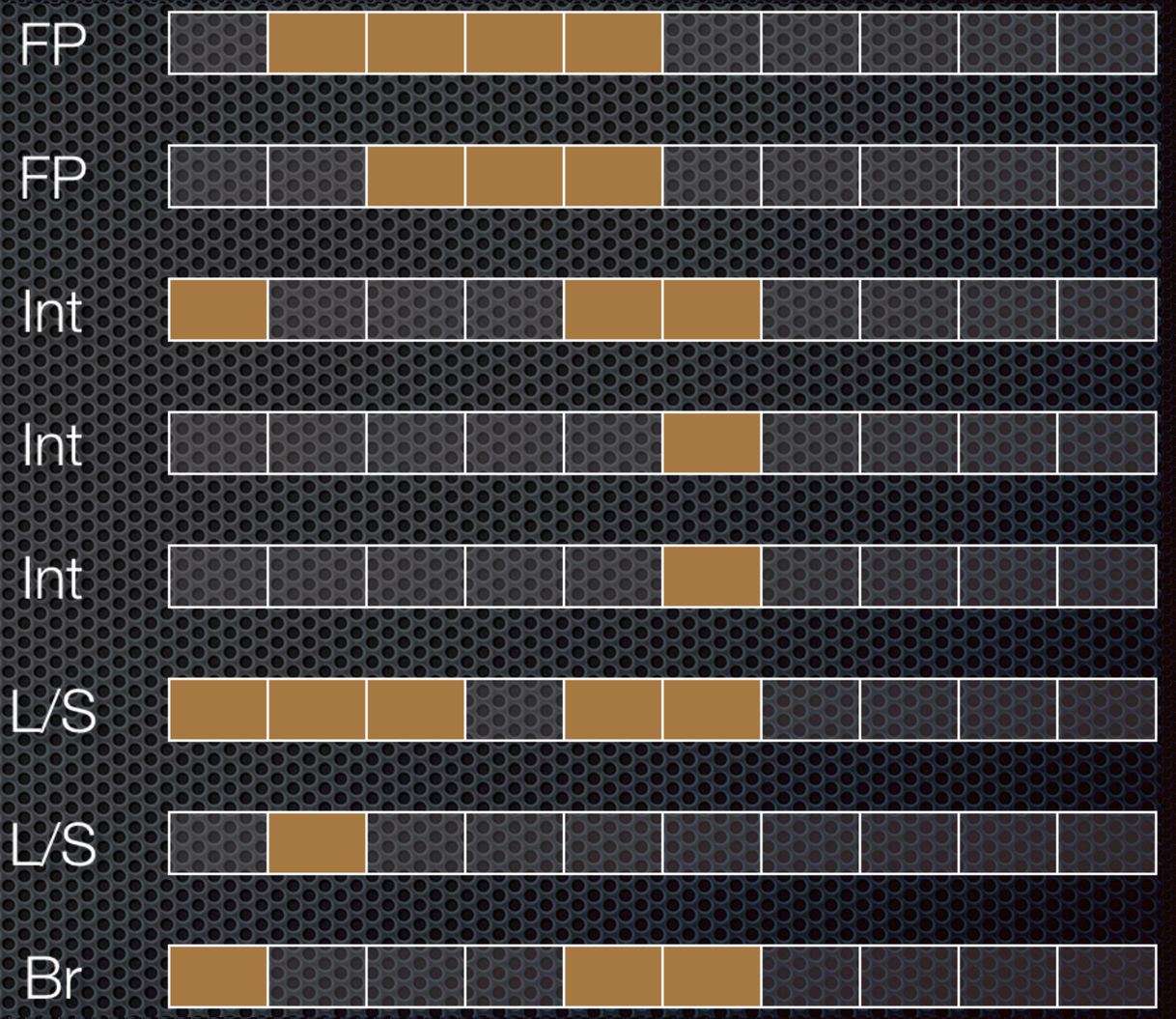

L/S
L/S
L/S
FP
FP
L/S
FP
FP
FP
Br
FP
FP
Int
Int
Int
L/S
L/S
Br

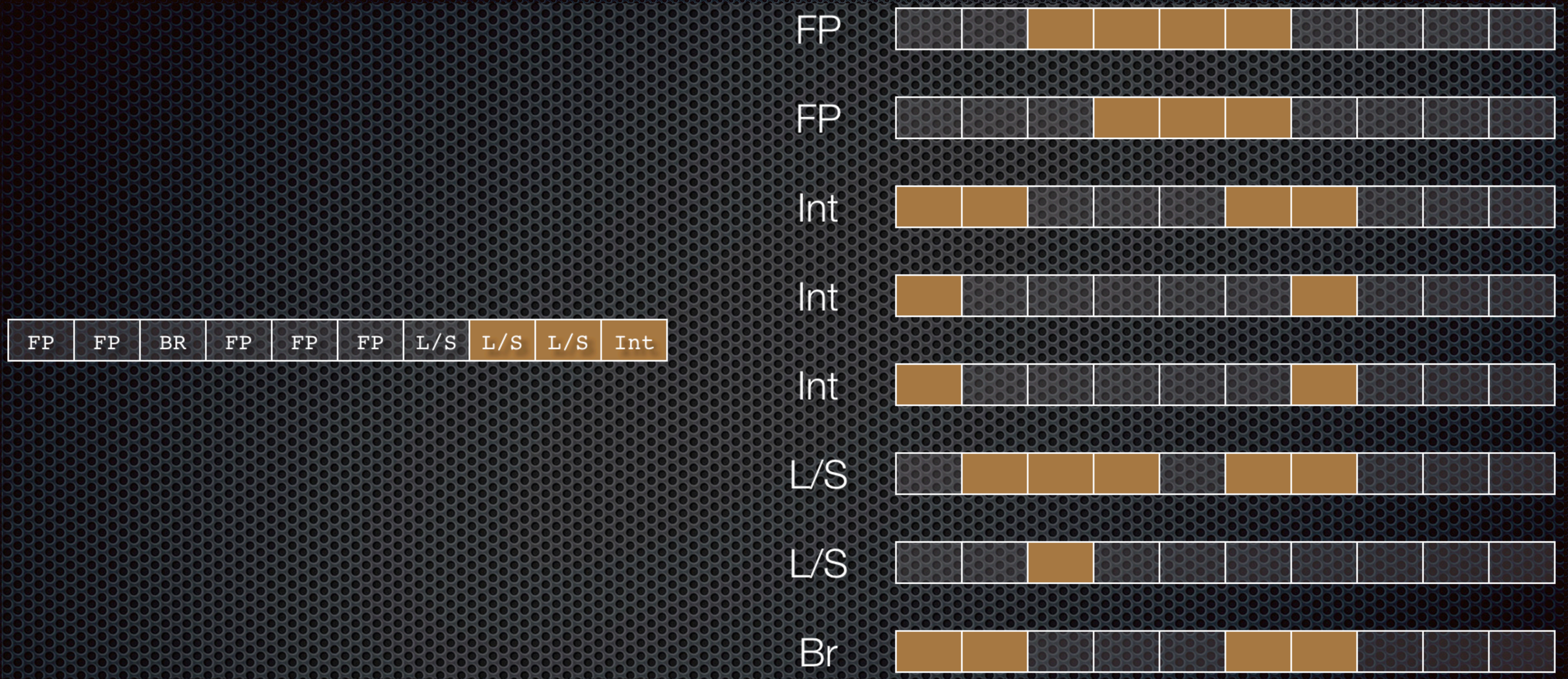

Br
Int
Br
L/S
L/S
L/S
FP
FP
L/S
FP
FP
FP
Int
Int
Int
L/S
L/S
Br


Int
Int
Int
Br
Int
Br
L/S
L/S
L/S
FP
FP
FP
Int
Int
Int
L/S
L/S
Br

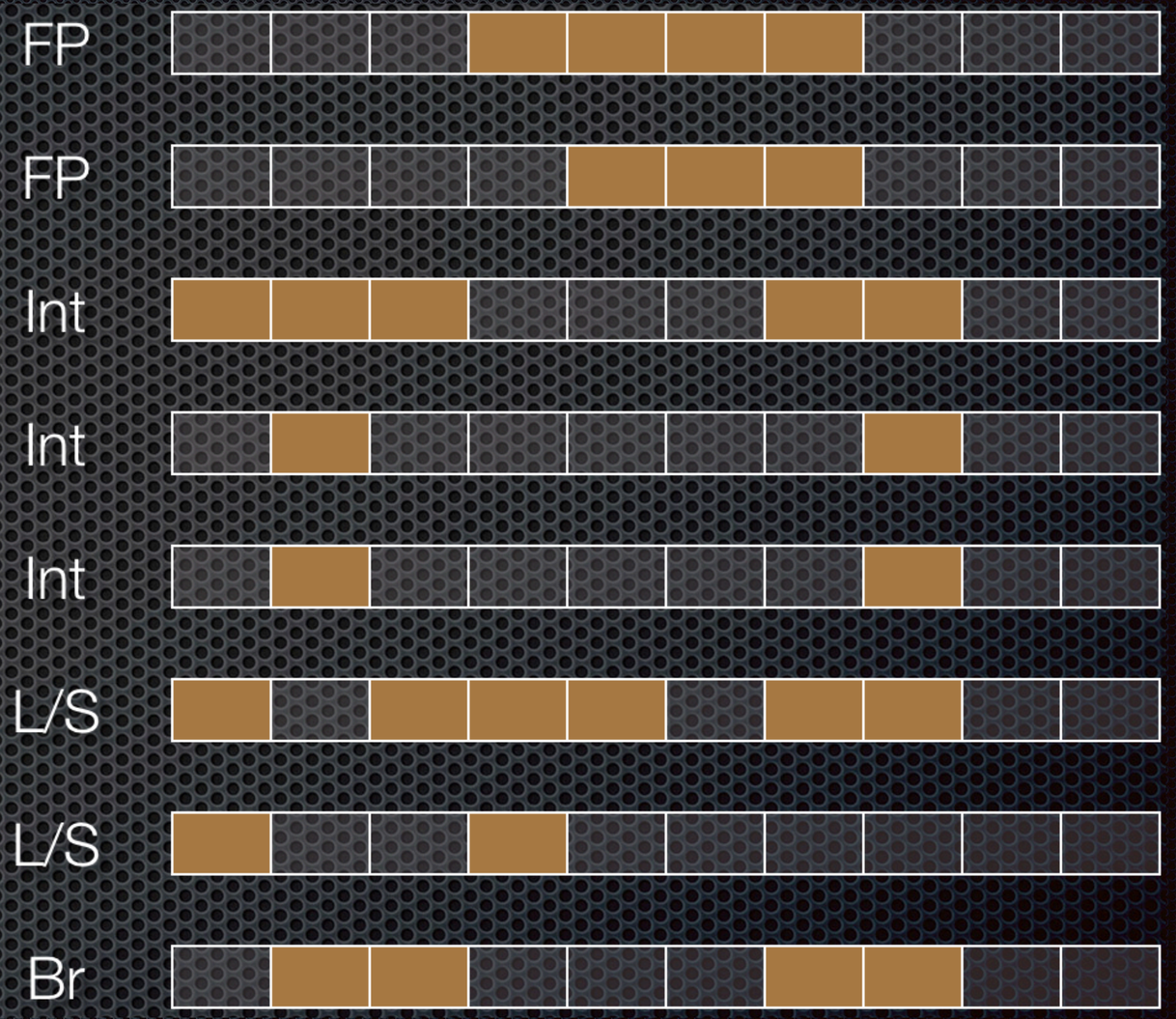

L/S
L/S
Int
Int
Int
Int
Br
Int
Br
L/S
FP
FP
Int
Int
Int
L/S
L/S
Br

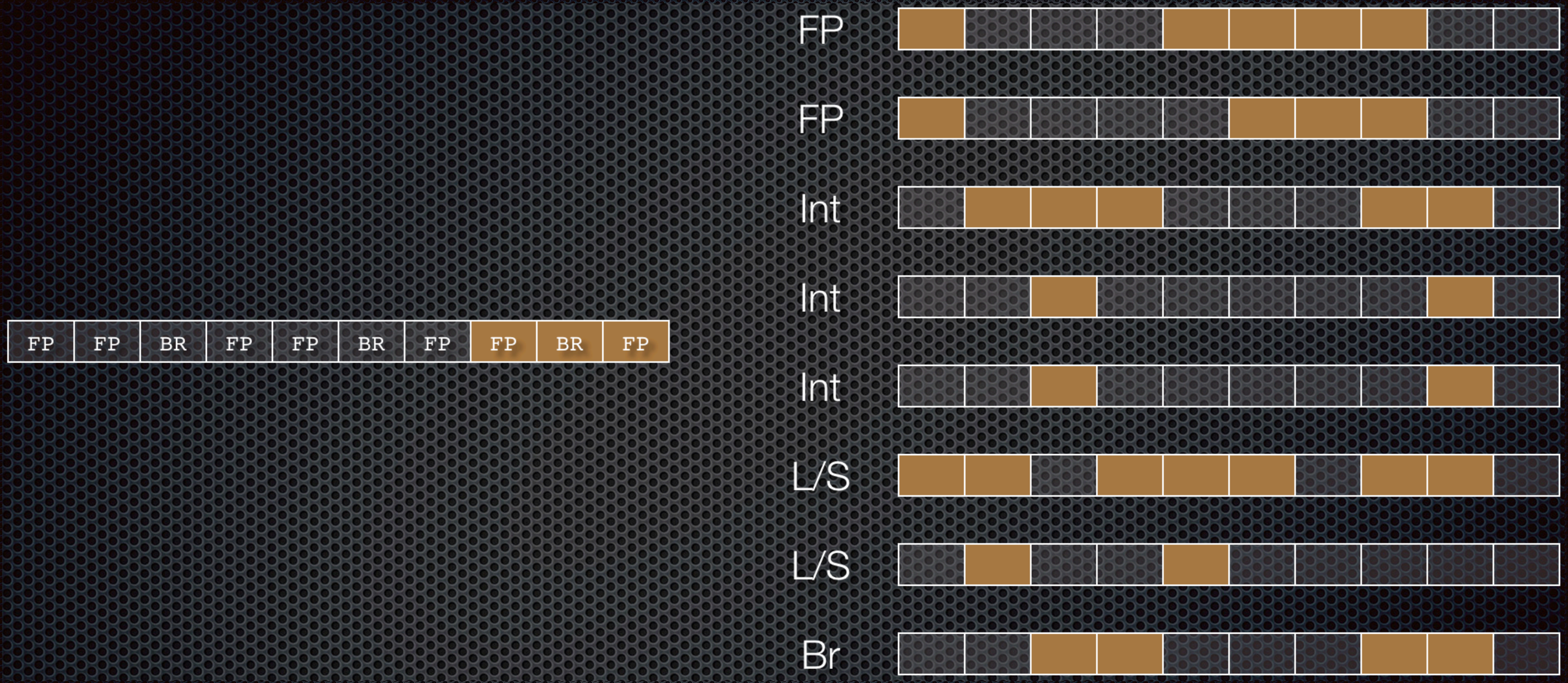
FP	FP	L/S	L/S	L/S	Int	Int	Int	Int	Br

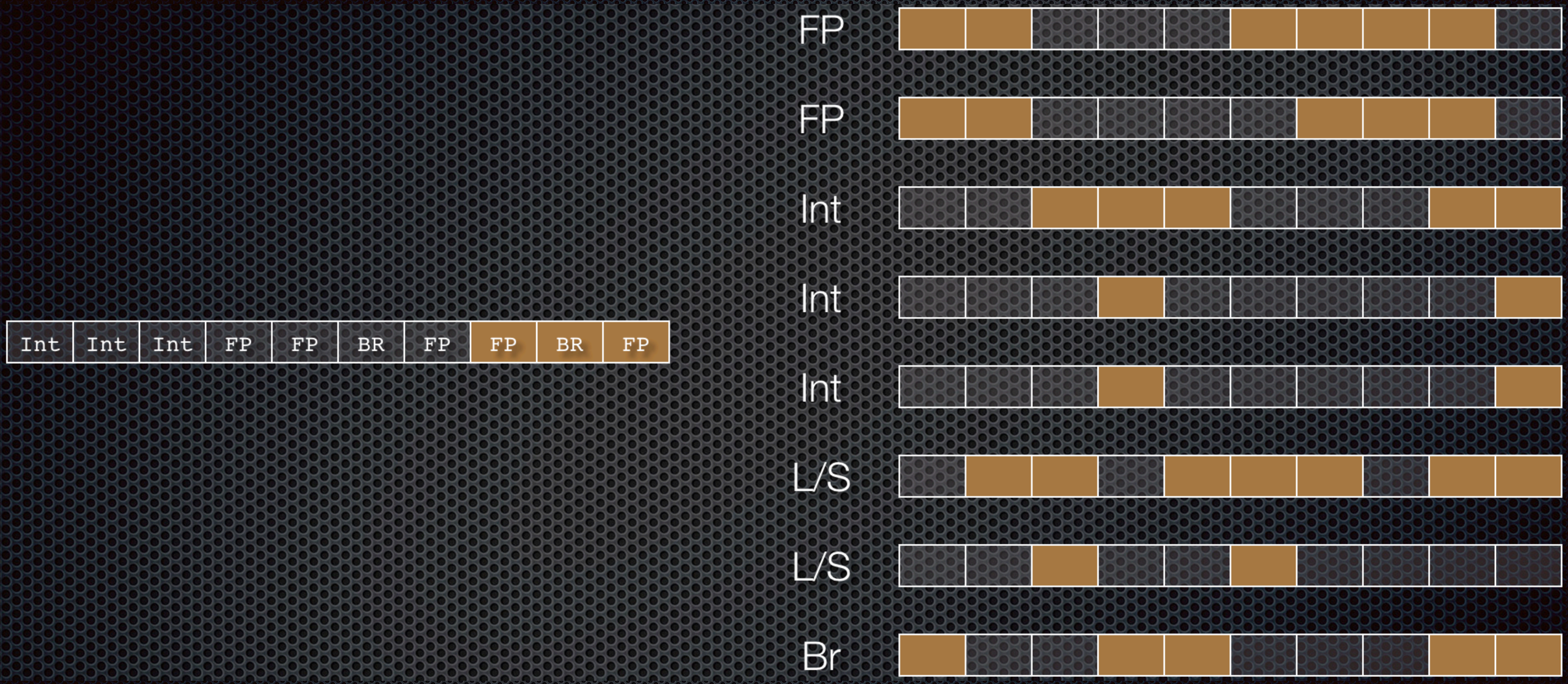



FP
FP
BR
FP
FP
FP
L/S
L/S
L/S
Int
FP
FP
Int
Int
Int
L/S
L/S
Br


FP
FP
BR
FP
FP
BR
FP
FP
FP
L/S


FP
FP
Int
Int
Int
L/S
L/S
Br

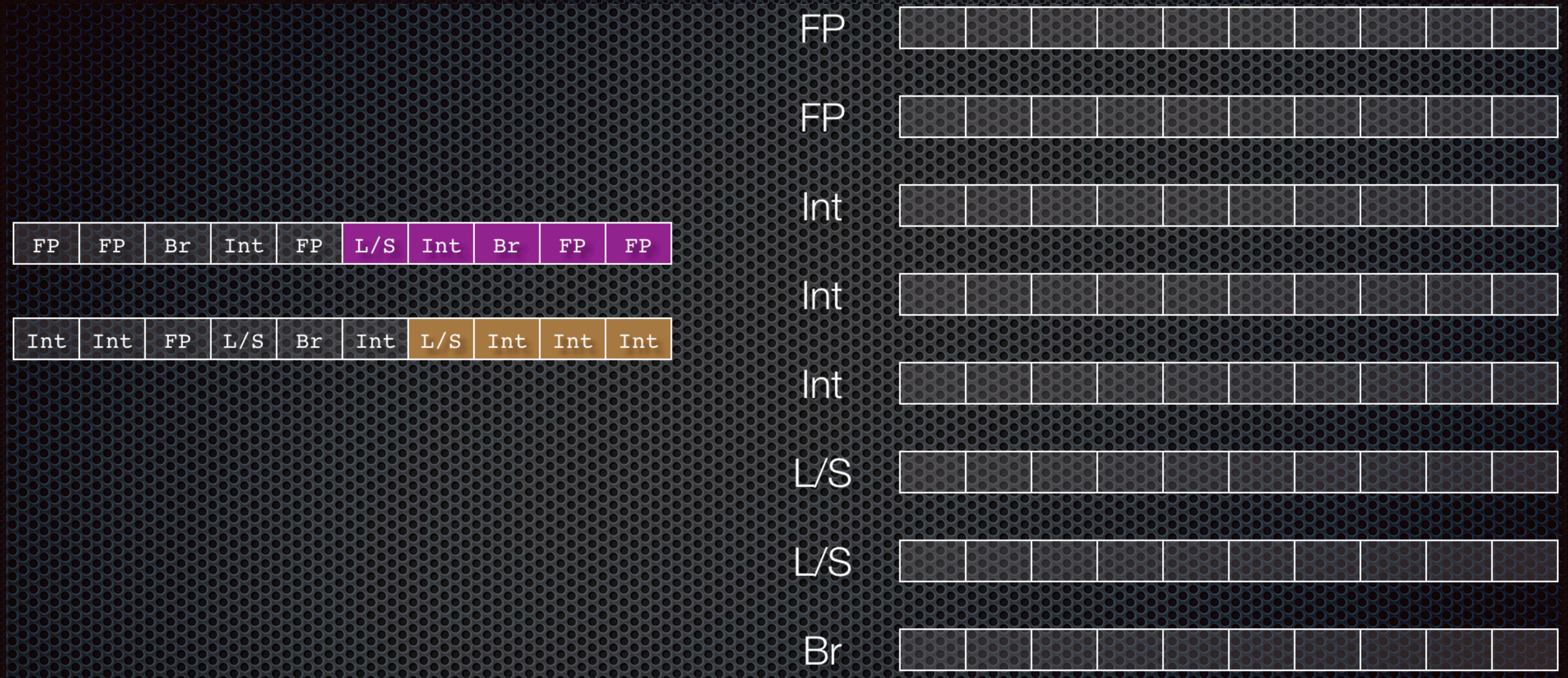

FP
FP
Int
Int
Int
L/S
L/S
Br
FP FP BR FP FP BR FP FP BR FP

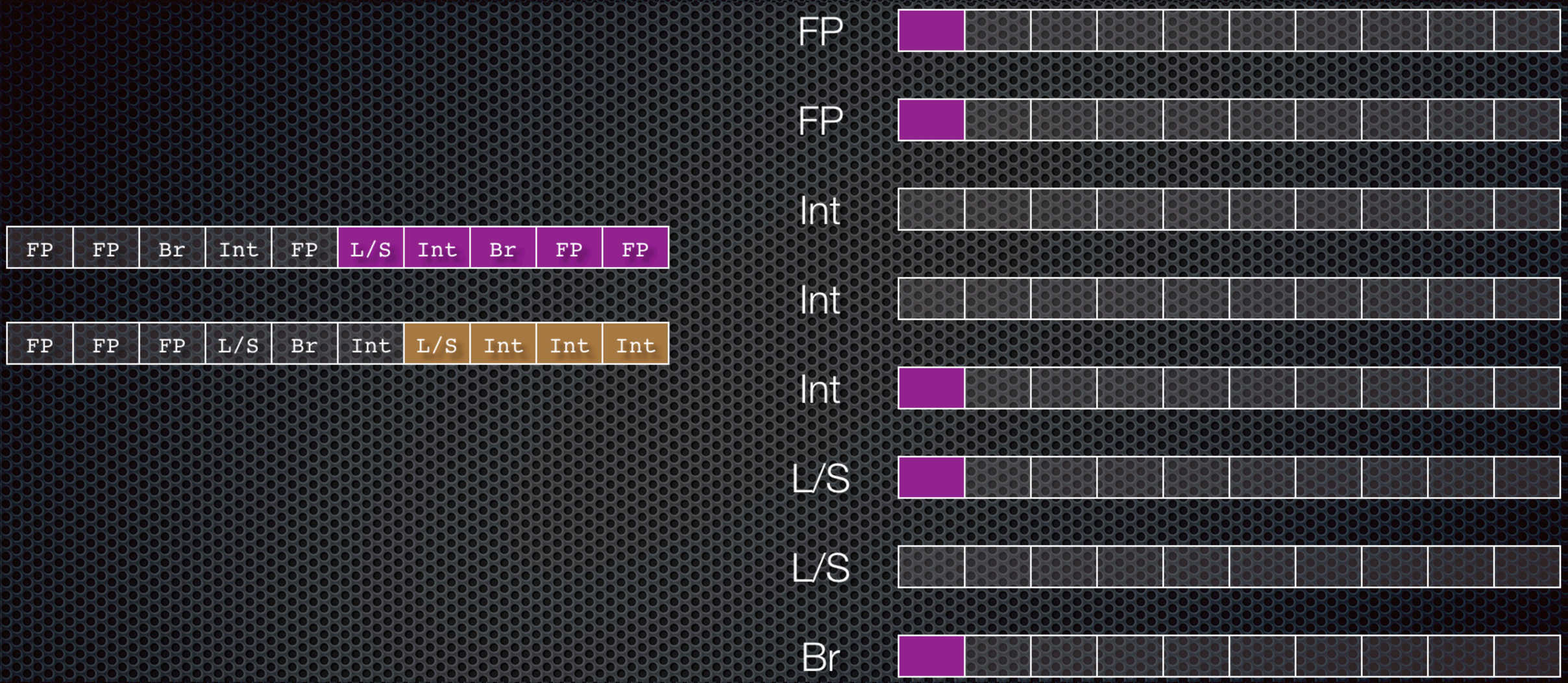


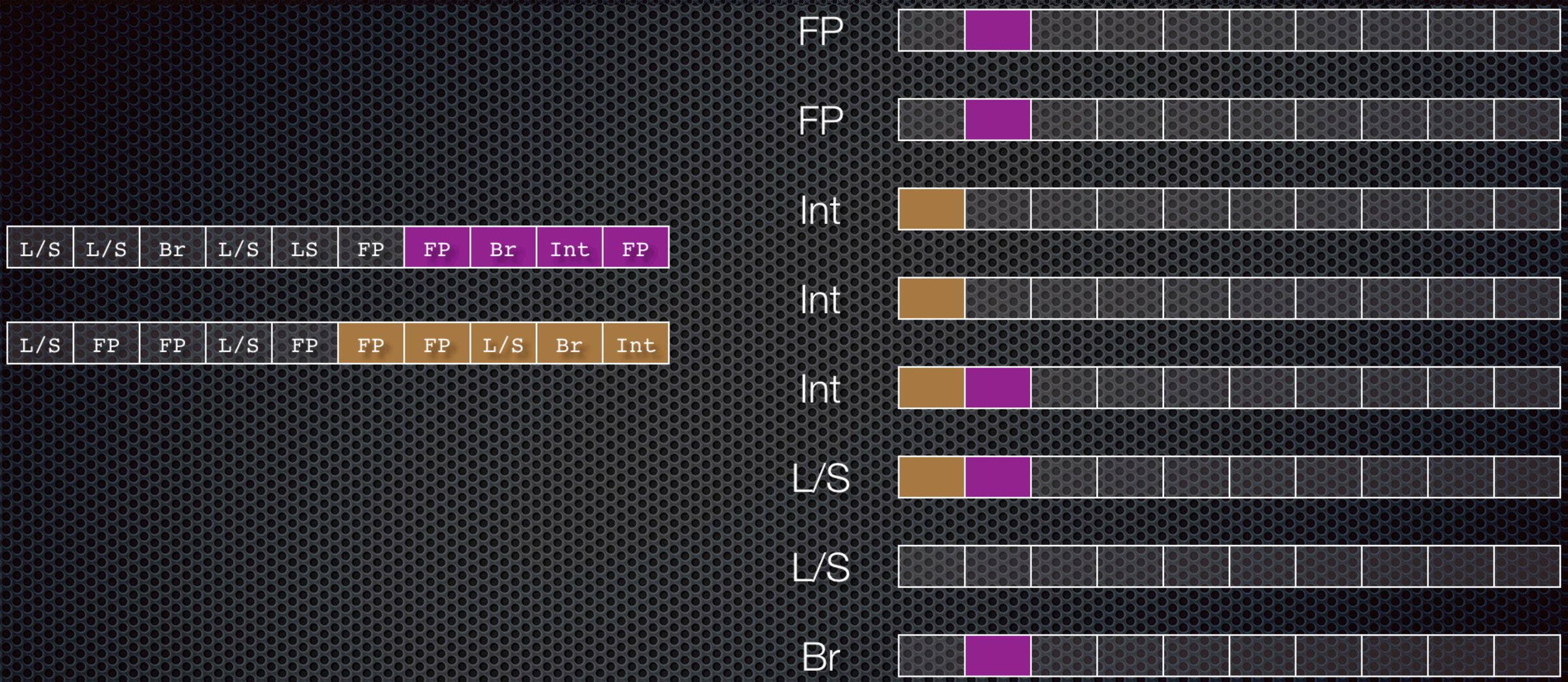
34 slots filled/80 available = 42.5% utilization

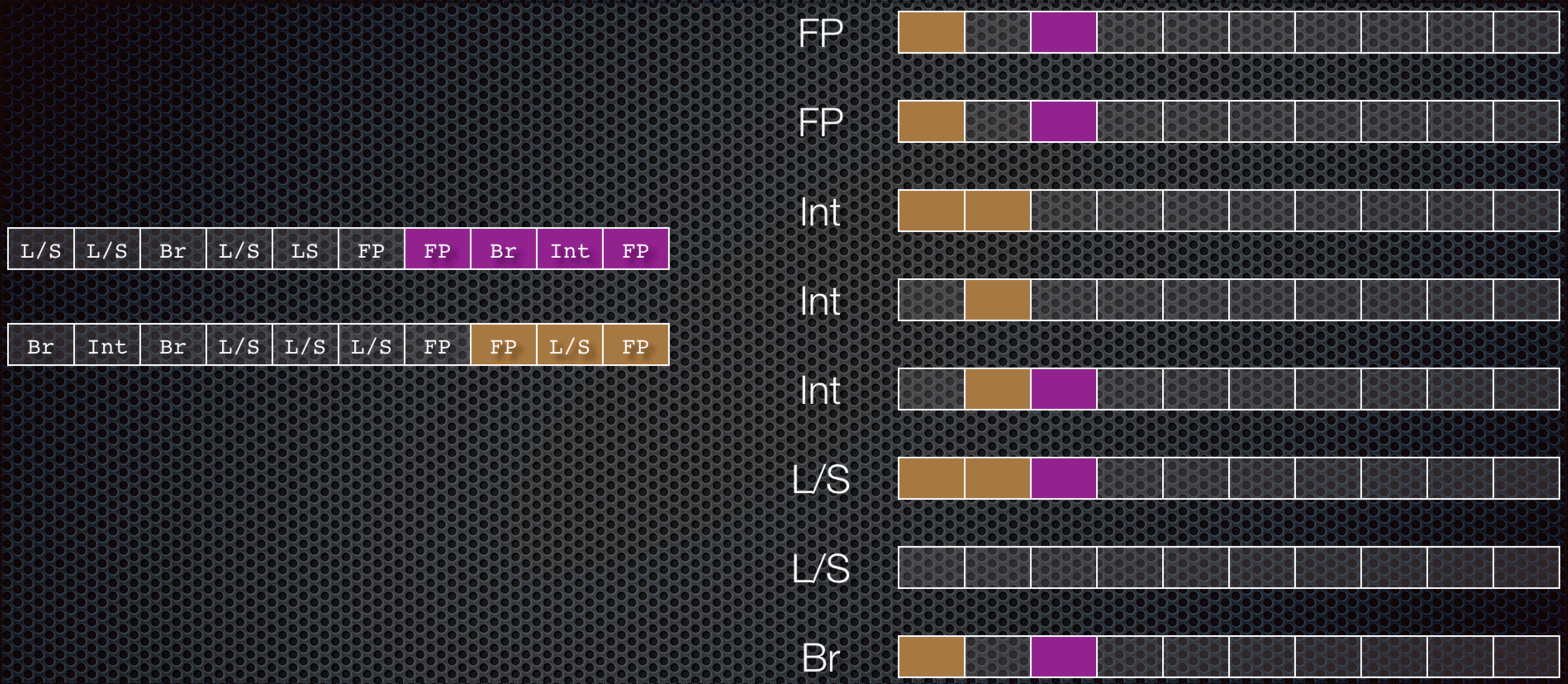
Simple Multithreading

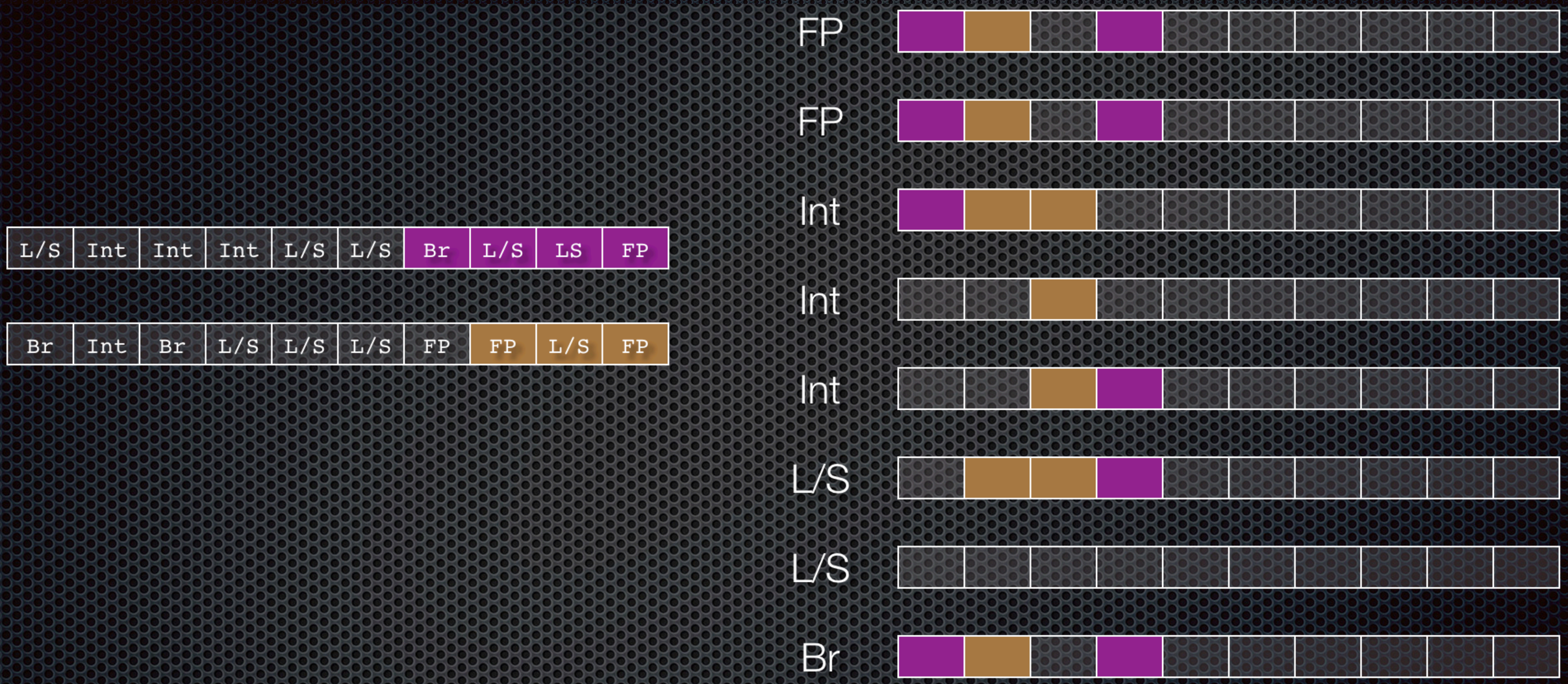
- Prefetch two threads at once
- Simplest source is user and supervisor
- Issue from thread with most ready ops
- Do not issue from stalled thread

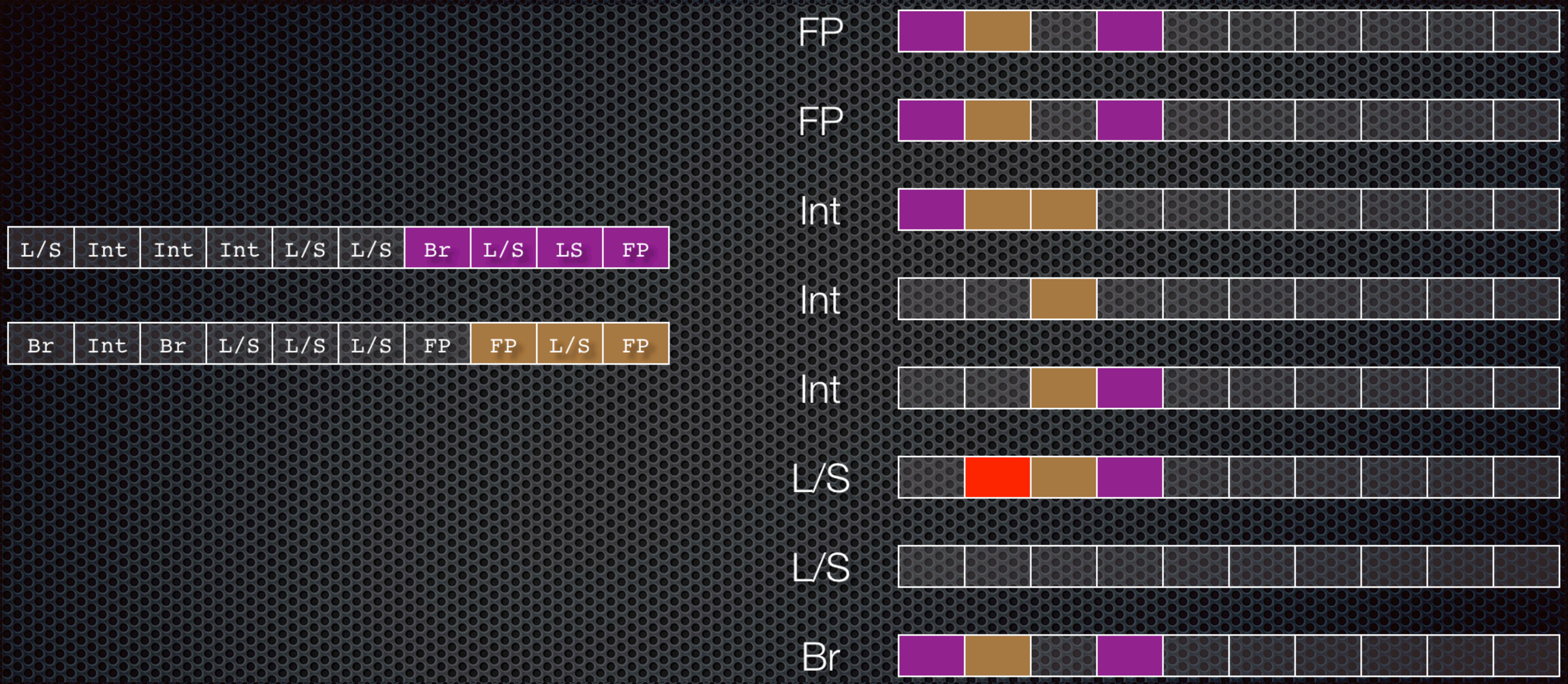

FP
FP
Br
Int
FP
L/S
Int
Br
FP
FP
Int
Int
FP
L/S
Br
Int
L/S
Int
Int
Int
FP
FP
Int
Int
Int
L/S
L/S
Br


FP
FP
Int
Int
Int
L/S
L/S
Br
FP FP Br Int FP L/S Int Br FP FP
FP FP FP L/S Br Int L/S Int Int Int


L/S
L/S
Br
L/S
LS
FP
FP
Br
Int
FP
L/S
FP
FP
L/S
FP
FP
FP
L/S
Br
Int
FP
FP
Int
Int
Int
L/S
L/S
Br


L/S
L/S
Br
L/S
LS
FP
FP
Br
Int
FP
Br
Int
Br
L/S
L/S
L/S
FP
FP
L/S
FP
FP
FP
Int
Int
Int
L/S
L/S
Br

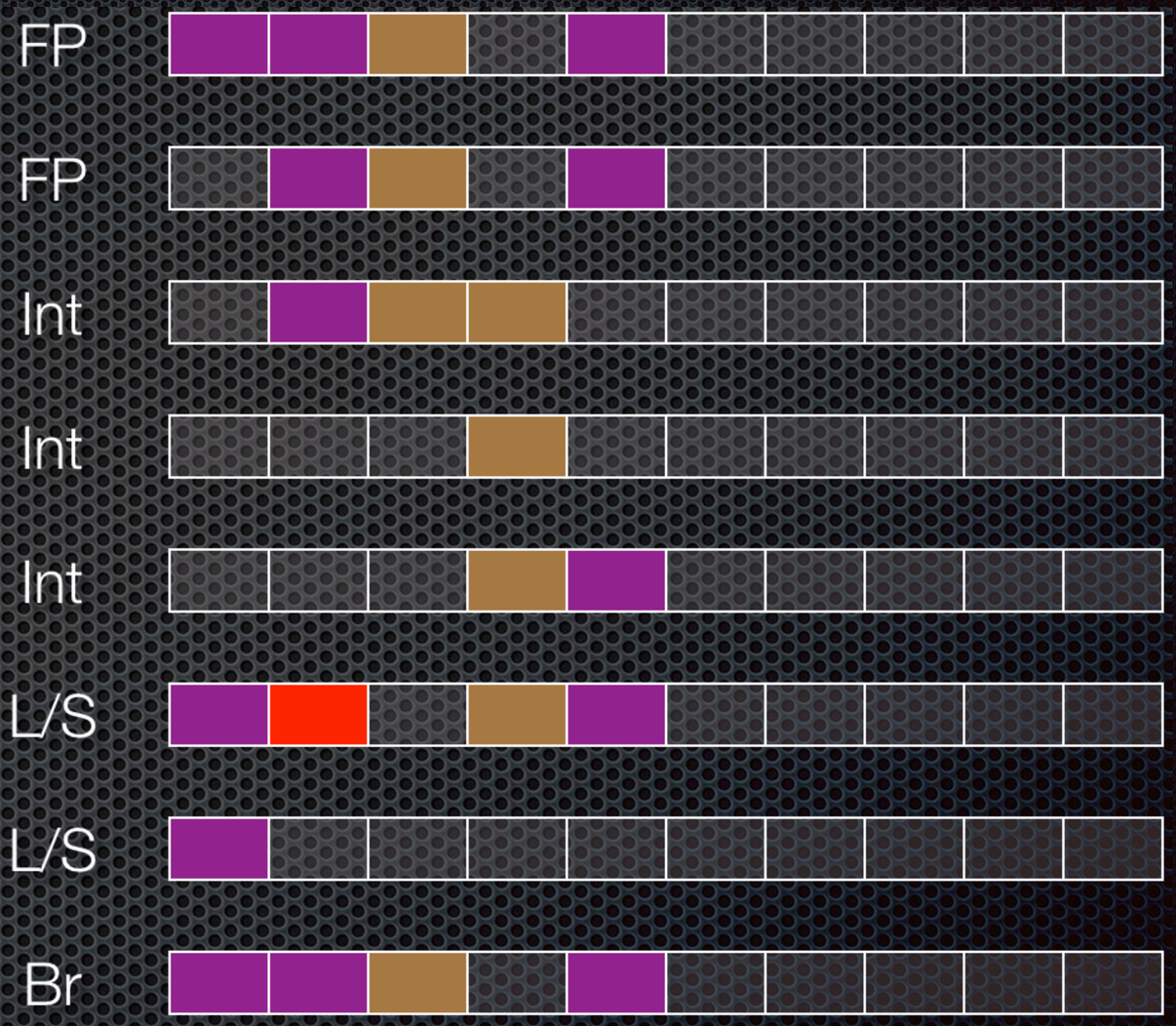

L/S
Int
Int
Int
L/S
L/S
Br
L/S
LS
FP
Br
Int
Br
L/S
L/S
L/S
FP
FP
L/S
FP
FP
FP
Int
Int
Int
L/S
L/S
Br



Assume the red L/S stalls 4 cycles

L/S	Int	Int	Int	L/S	Int	Int	Int	L/S	L/S

Br	Int	Br	L/S	L/S	L/S	FP	FP	L/S	FP

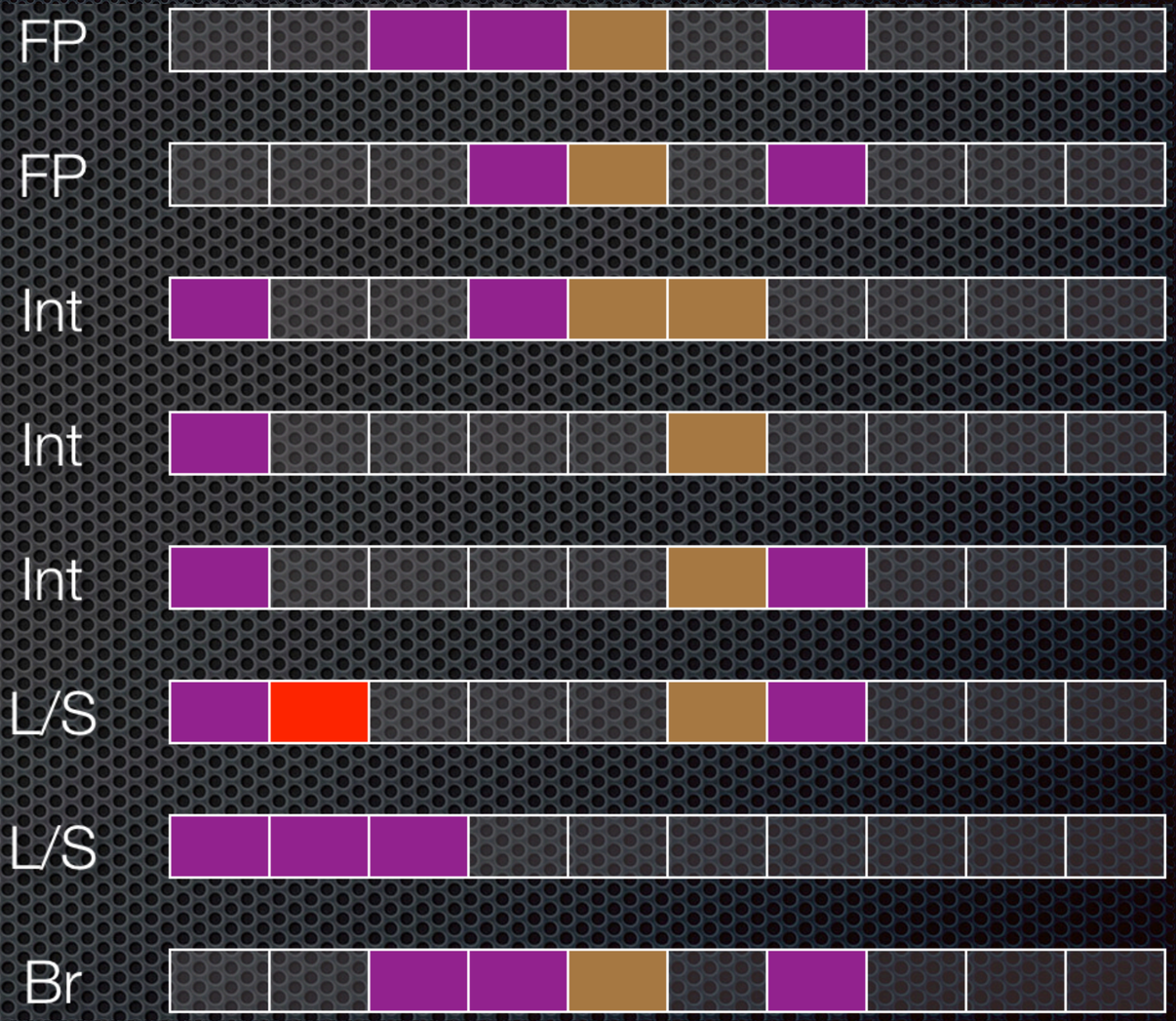


Br	L/S	Int	Int	Int	L/S	Int	Int	Int	L/S

Br	Int	Br	L/S	L/S	L/S	FP	FP	L/S	FP

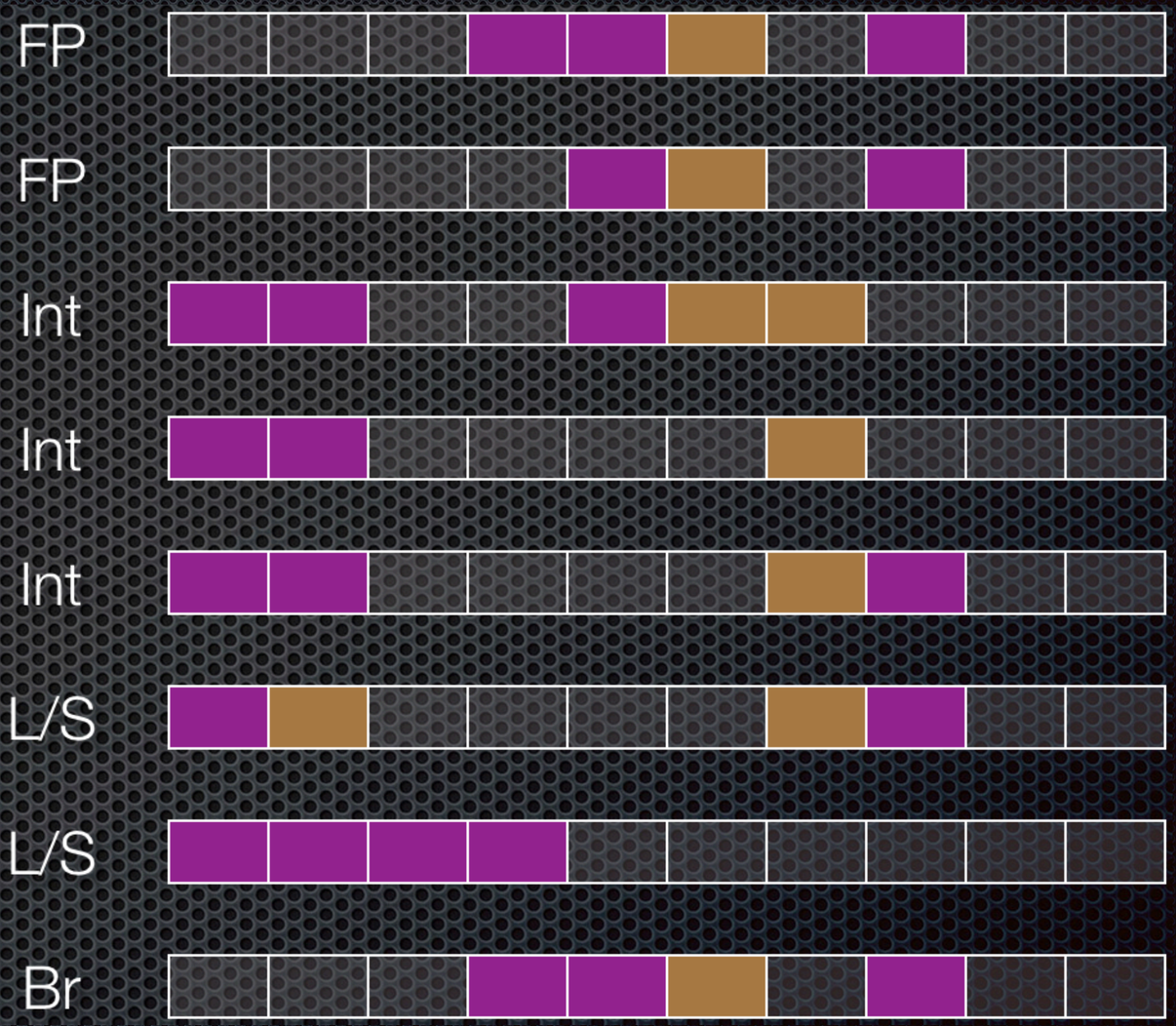
L/S	Int	FP	FP	Br	L/S	Int	Int	Int	L/S

Br	Int	Br	L/S	L/S	L/S	FP	FP	L/S	FP



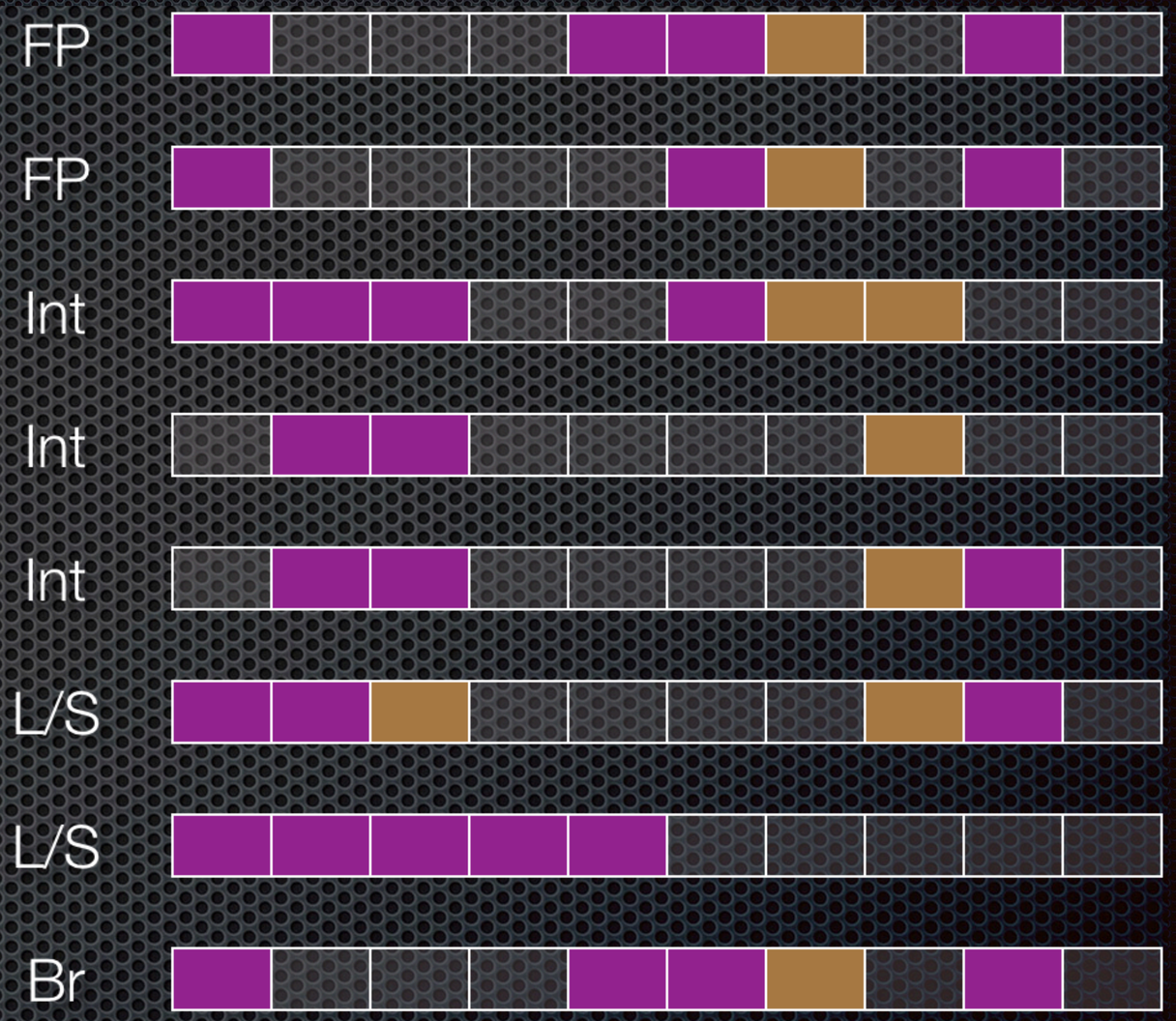
Int	L/S	L/S	L/S	L/S	Int	FP	FP	Br	L/S

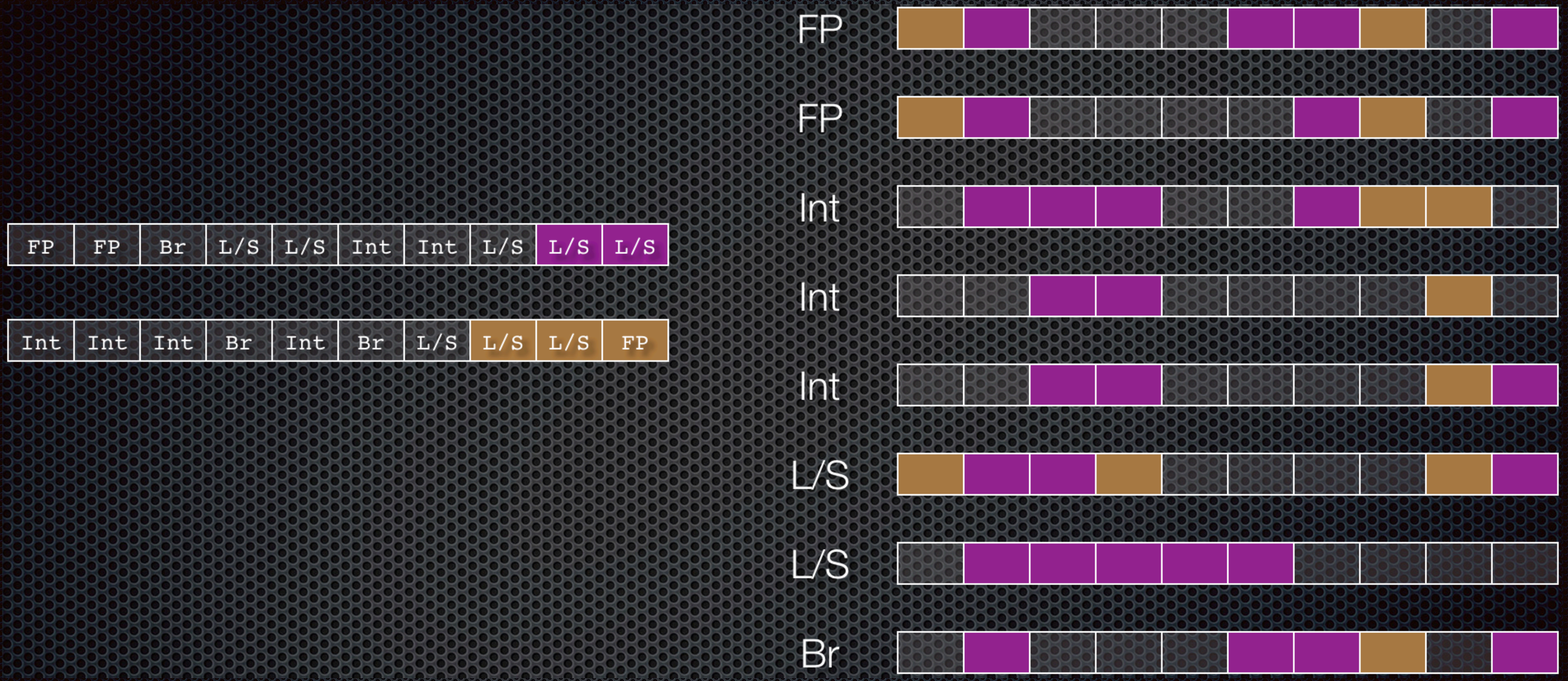
Br	Int	Br	L/S	L/S	L/S	FP	FP	L/S	FP



FP	FP	Br	L/S	L/S	Int	Int	L/S	L/S	L/S

Br	Int	Br	L/S	L/S	L/S	FP	FP	L/S	FP





40 slots filled/80 available = 50% utilization
(even with the extra overhead of a stall)

Notes

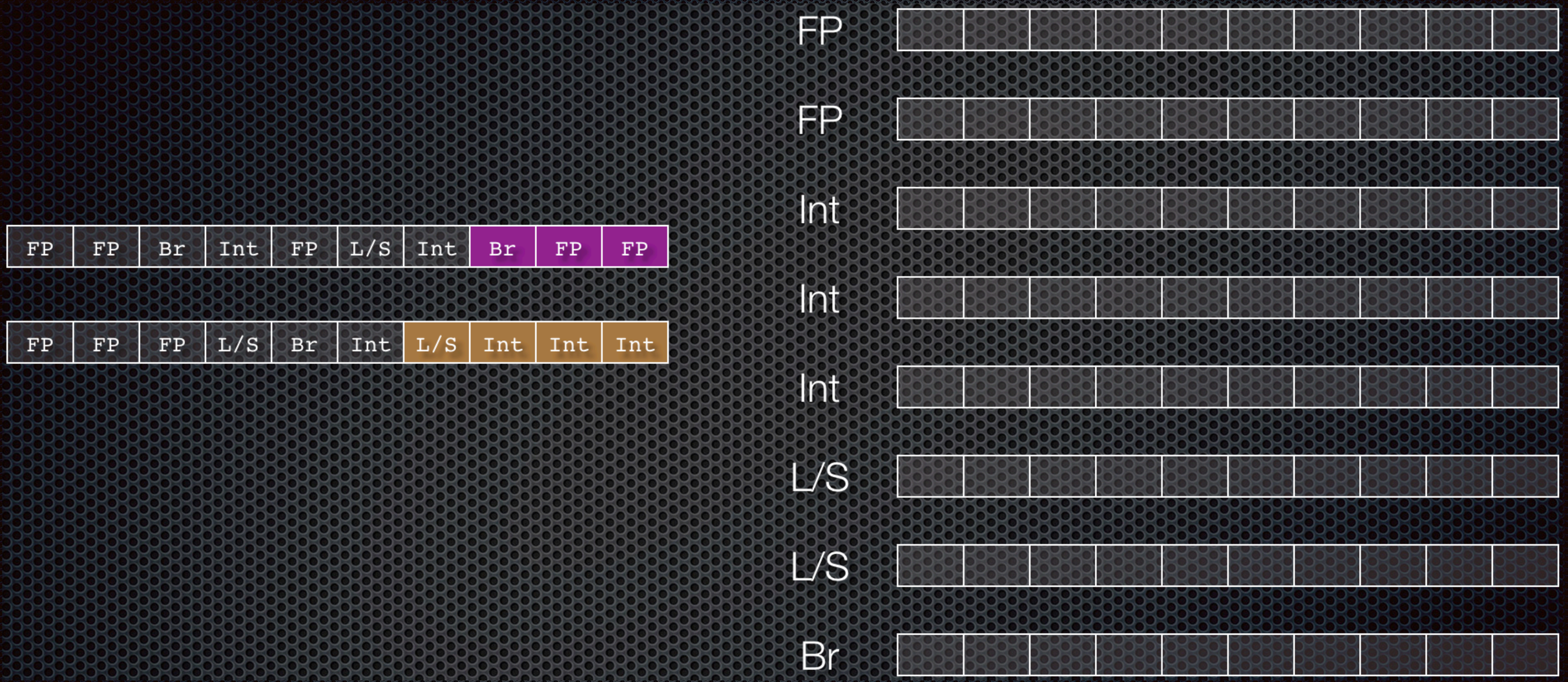
- Could have issued from both threads
- Likely that dependences in stalled thread would limit forward progress
- Some of boost in utilization was due to the second thread having a better schedule
- System threads rarely have FP ops
- Assumed no other hazards
- Real situation likely to have lower utilization

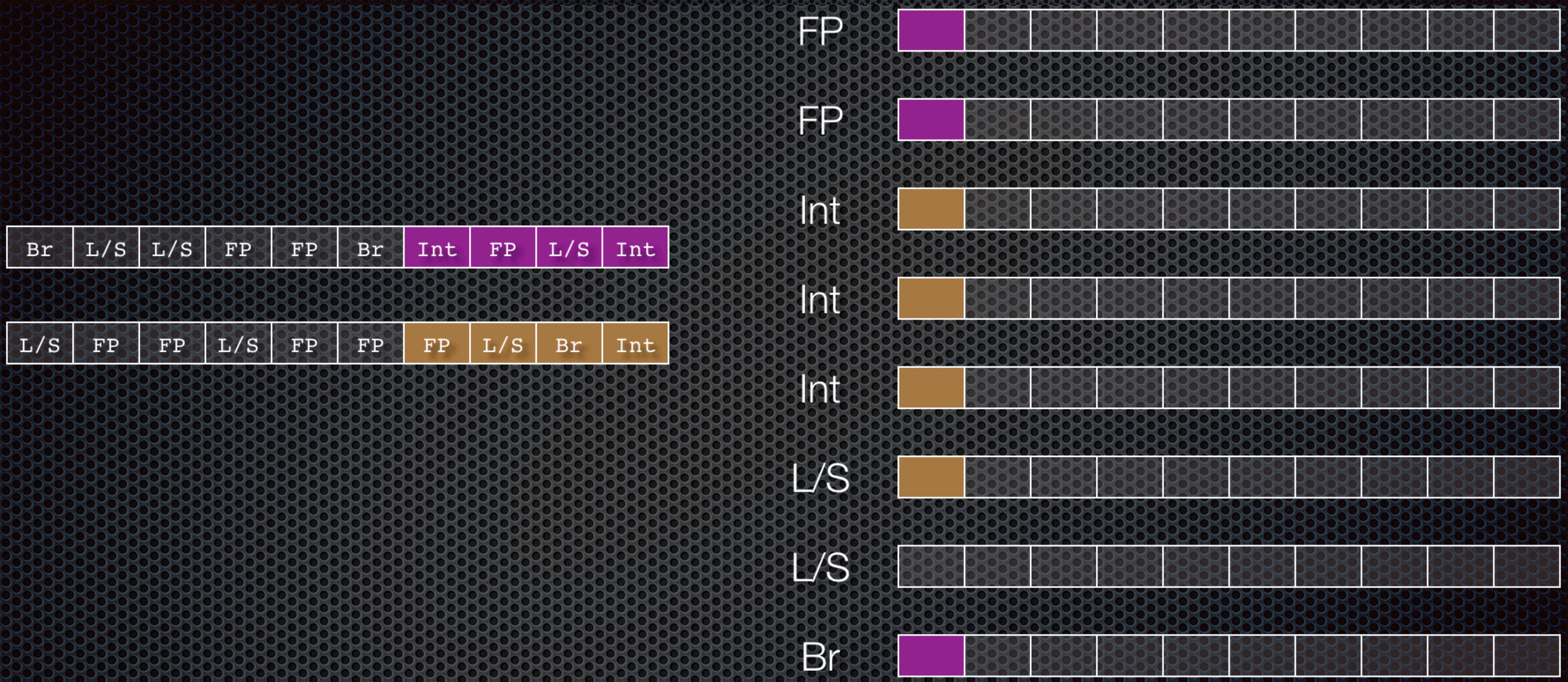
Cost of Multithreading

- Separate register bank(s) needed for each thread, including PC, PSW, etc.
- Switching logic to select register bank (and register update unit stations)
- Extra prefetch queue and logic per thread, more complex scheduling
- Thread tags for instructions in pipelines (more complex commit)
- Causes functional units to run hotter
- OS must manage threads

Simultaneous Multithreading

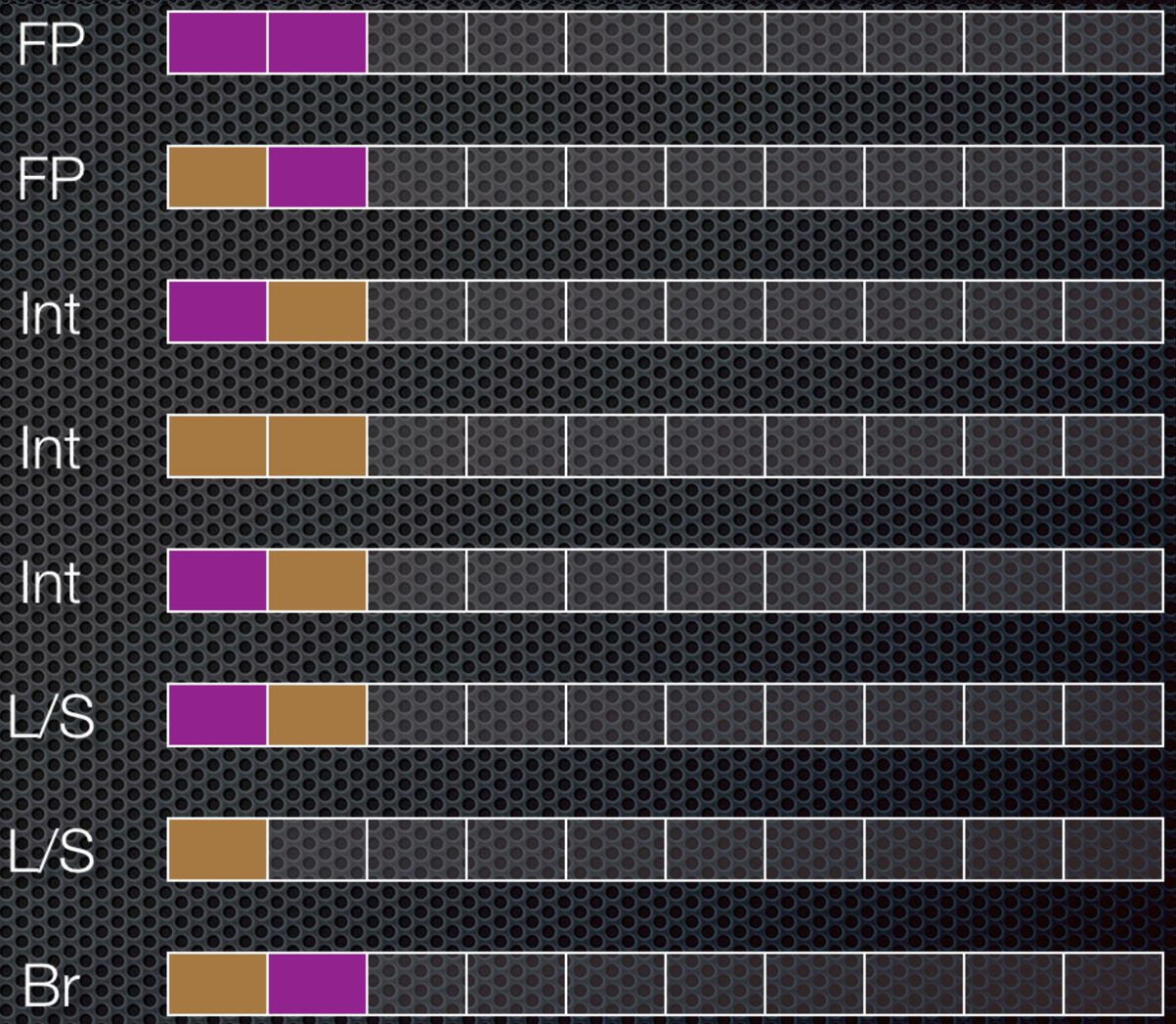
- Issue the maximum mix of ready ops from the two threads
- Requires analyzing a larger set of pending ops


FP
FP
Br
Int
FP
L/S
Int
Br
FP
FP
FP
FP
FP
L/S
Br
Int
L/S
Int
Int
Int
FP
FP
Int
Int
L/S
L/S
Br


Br L/S L/S FP FP Br Int FP L/S Int
L/S FP FP L/S FP FP FP L/S Br Int
FP
FP
Int
Int
Int
L/S
L/S
Br

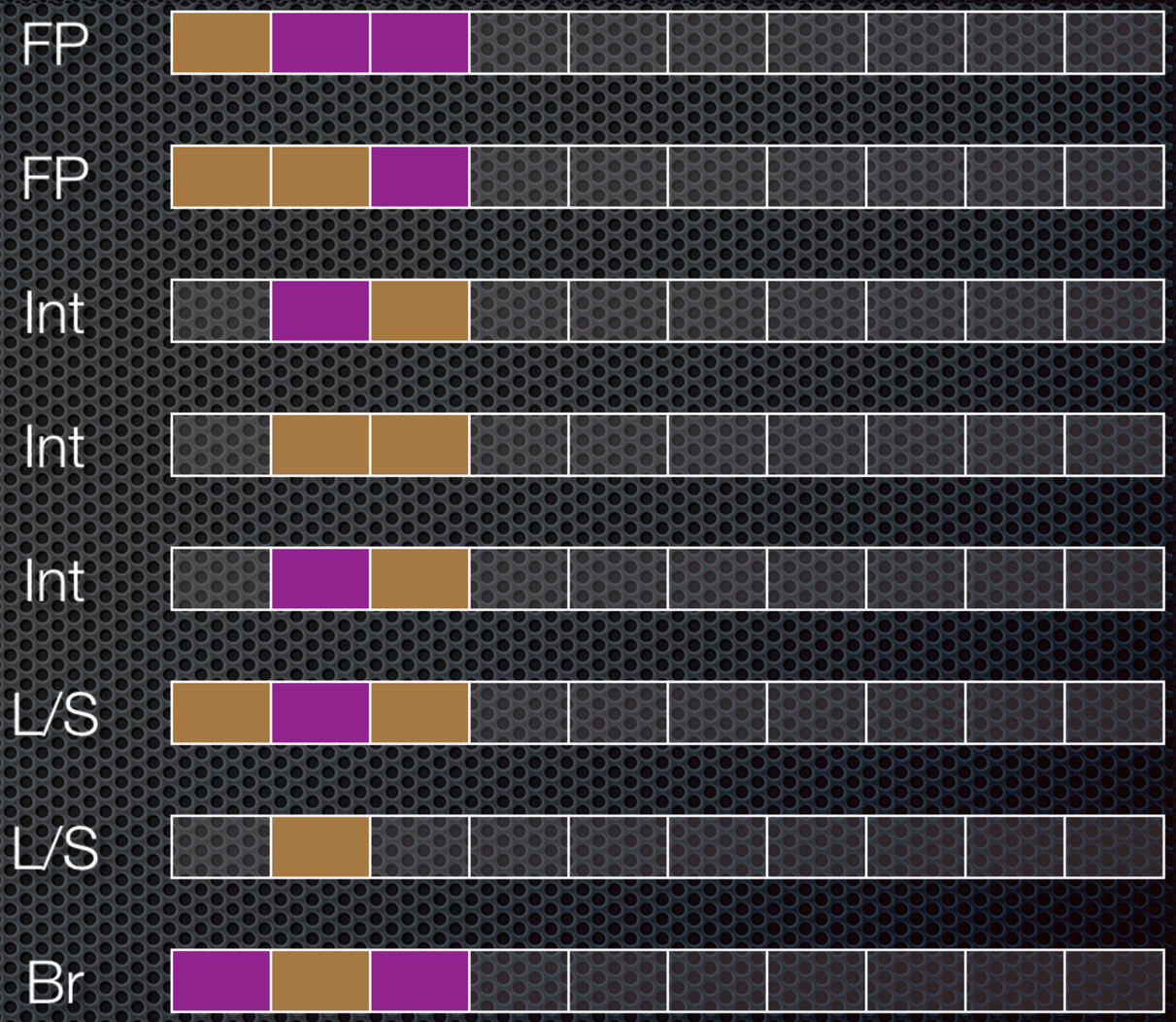
Int	Int	L/S	L/S	Br	L/S	L/S	FP	FP	Br

Int	Br	L/S	L/S	L/S	FP	FP	L/S	FP	FP



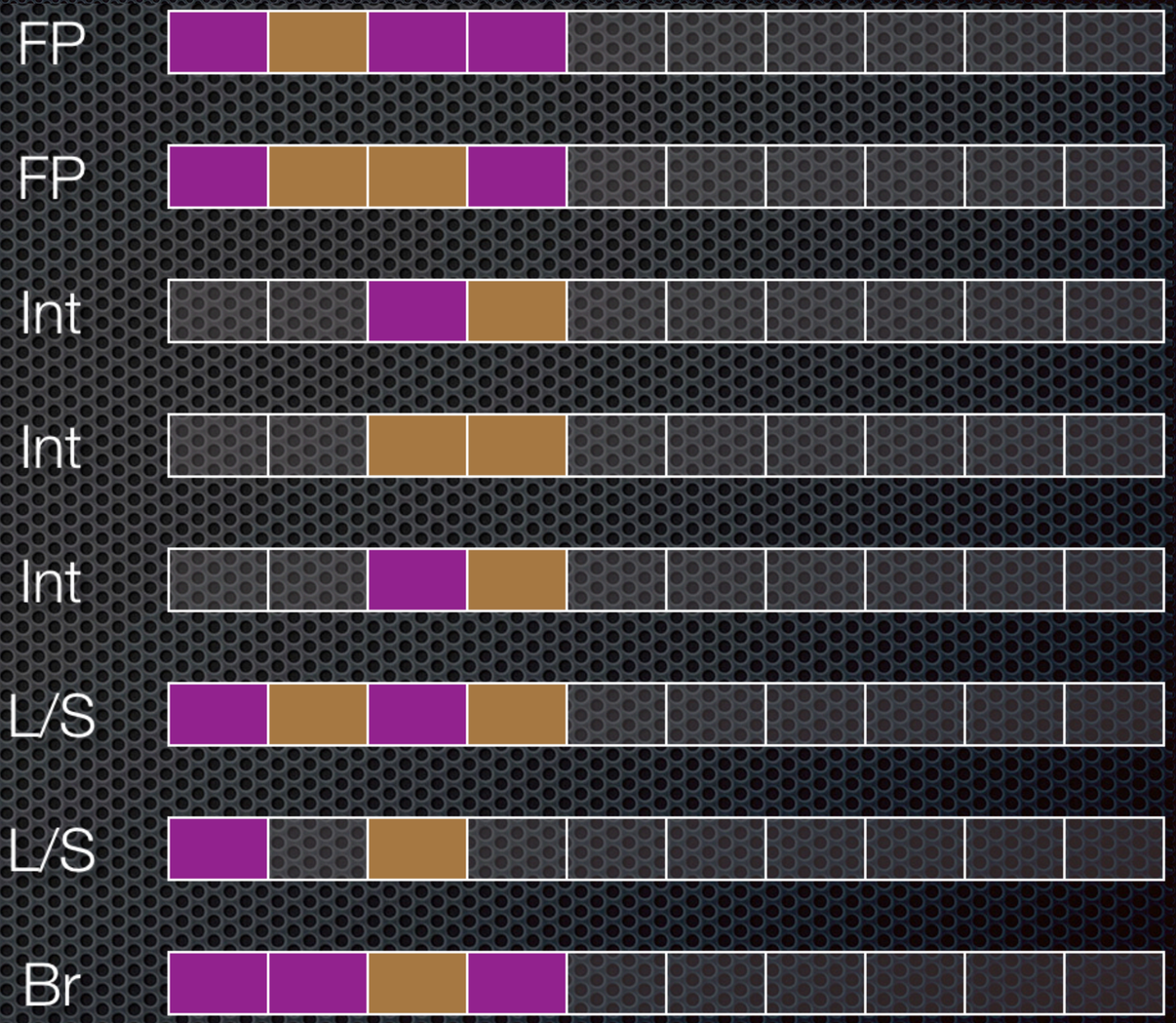
Int	Int	Int	L/S	L/S	Br	L/S	L/S	FP	FP

Int	Int	Br	Int	Br	L/S	L/S	L/S	FP	FP



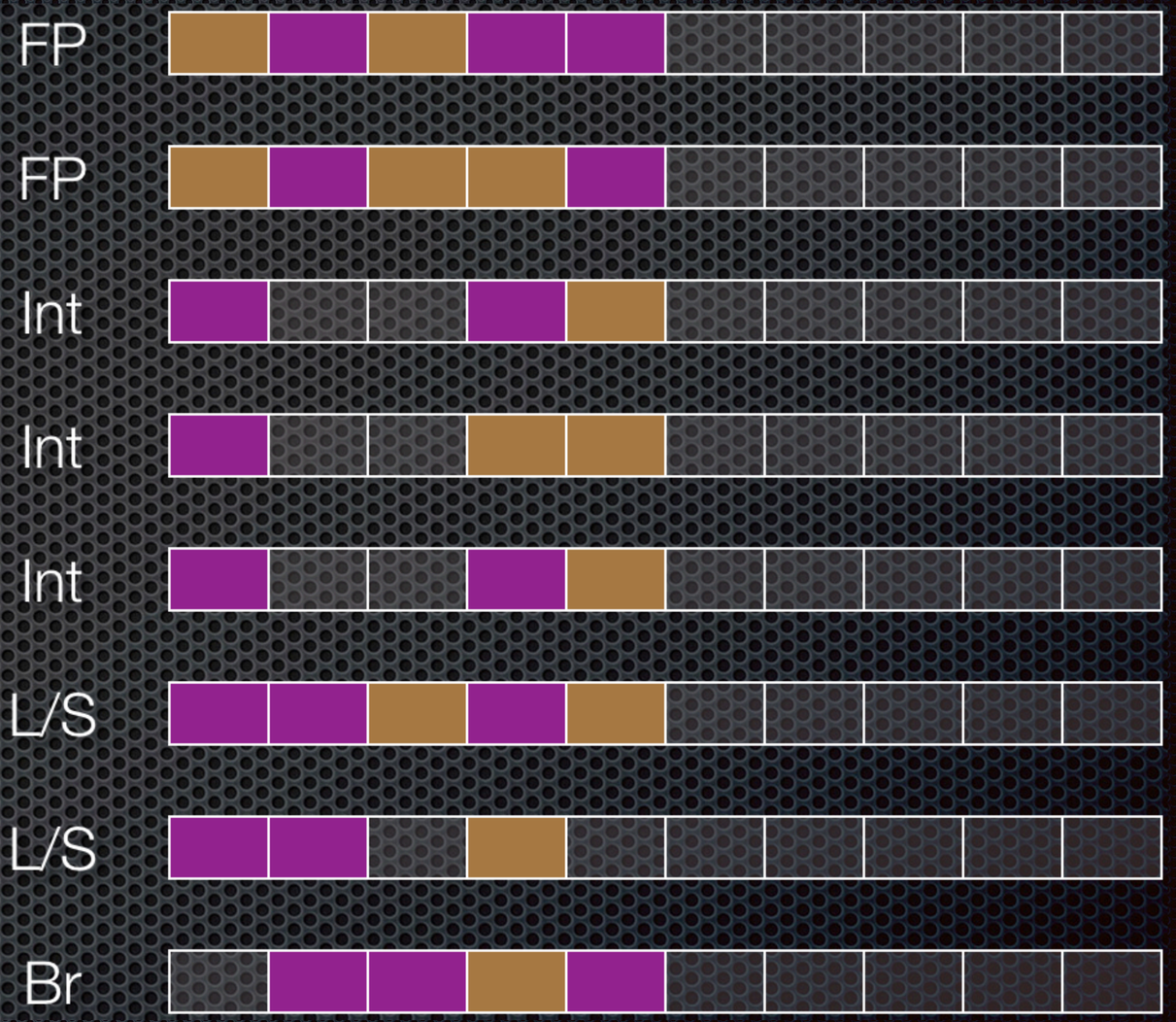
Int	Int	Int	L/S	Int	Int	Int	Int	L/S	L/S

Int	Int	Br	Int	Br	L/S	L/S	L/S	FP	FP



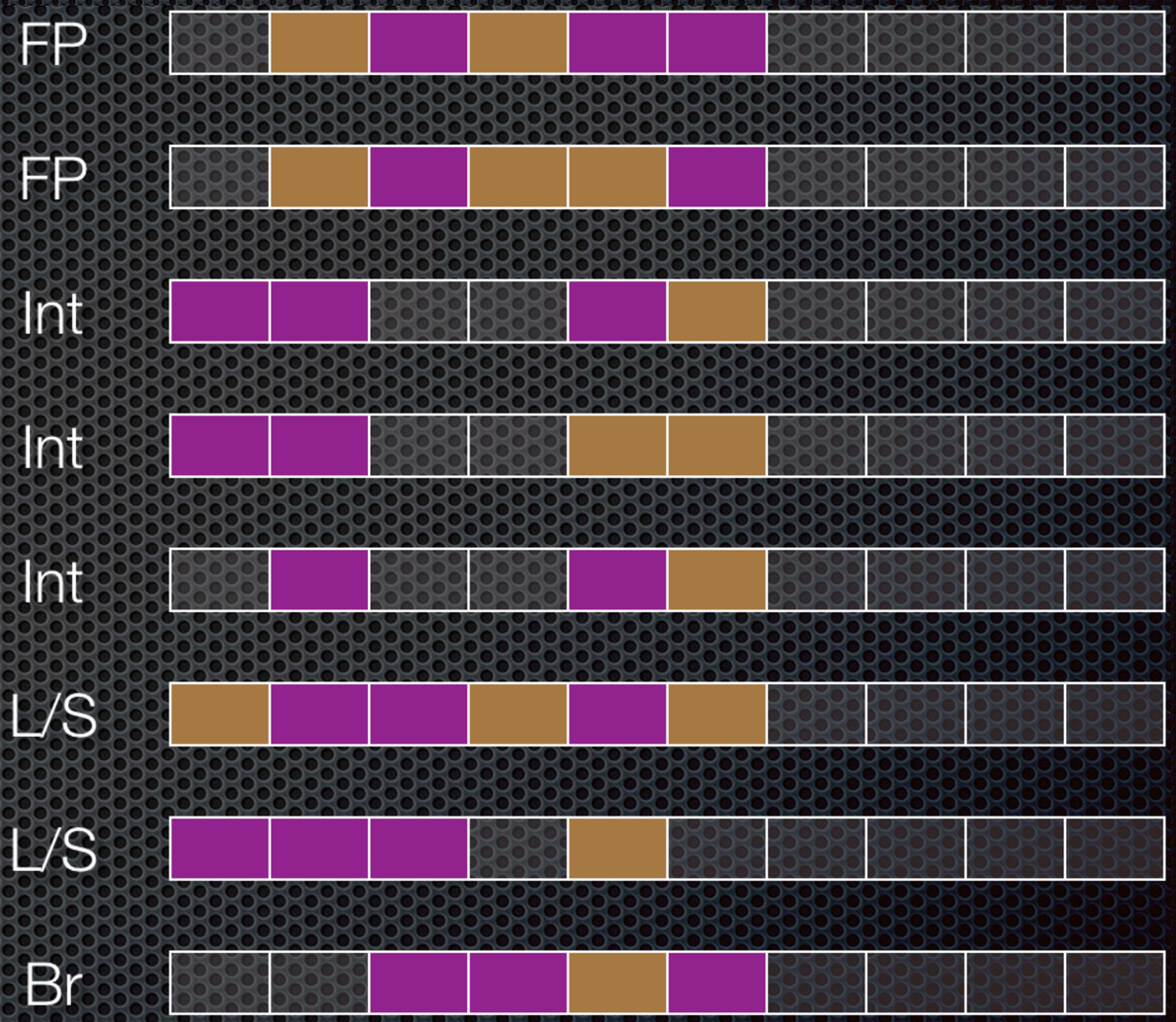
Int	FP	FP	Br	L/S	Int	Int	Int	L/S	Int

Int	Int	Int	Int	Br	Int	Br	L/S	L/S	L/S



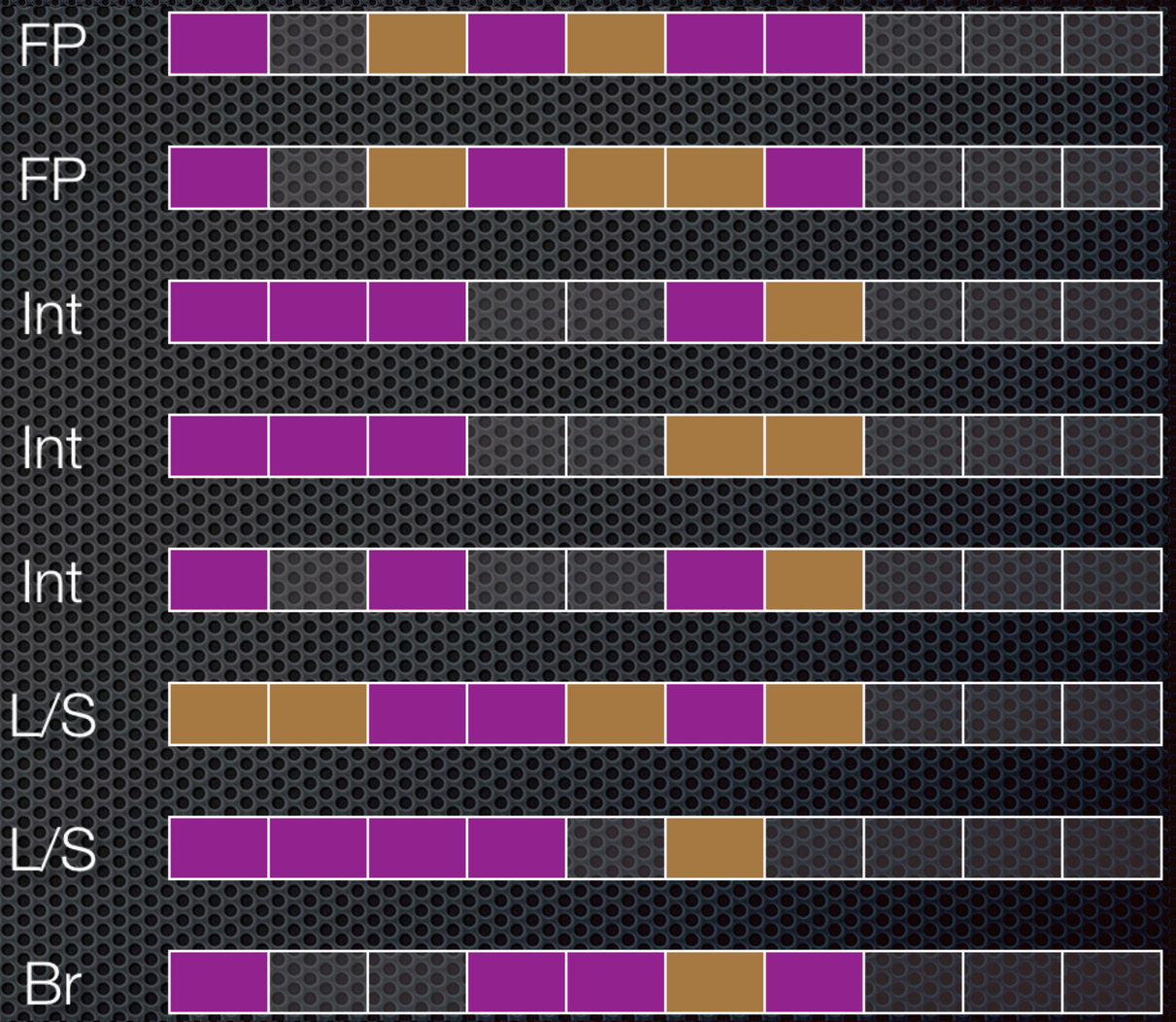
L/S	L/S	L/S	Int	FP	FP	Br	L/S	Int	Int

L/S	Int	Int	Int	Int	Br	Int	Br	L/S	L/S



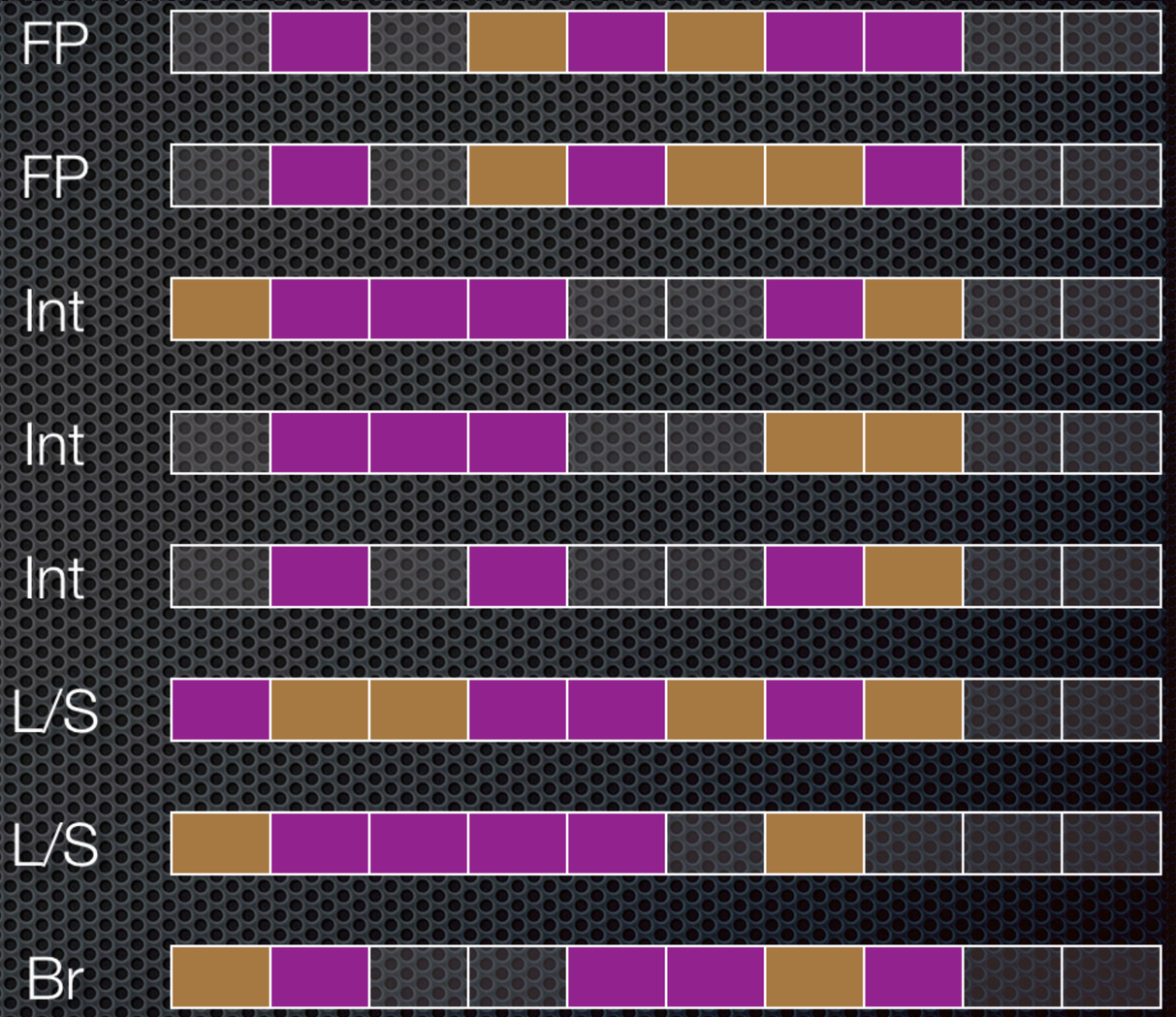
FP	Br	L/S	L/S	Int	Int	L/S	L/S	L/S	L/S

L/S	L/S	Int	Int	Int	Int	Br	Int	Br	L/S



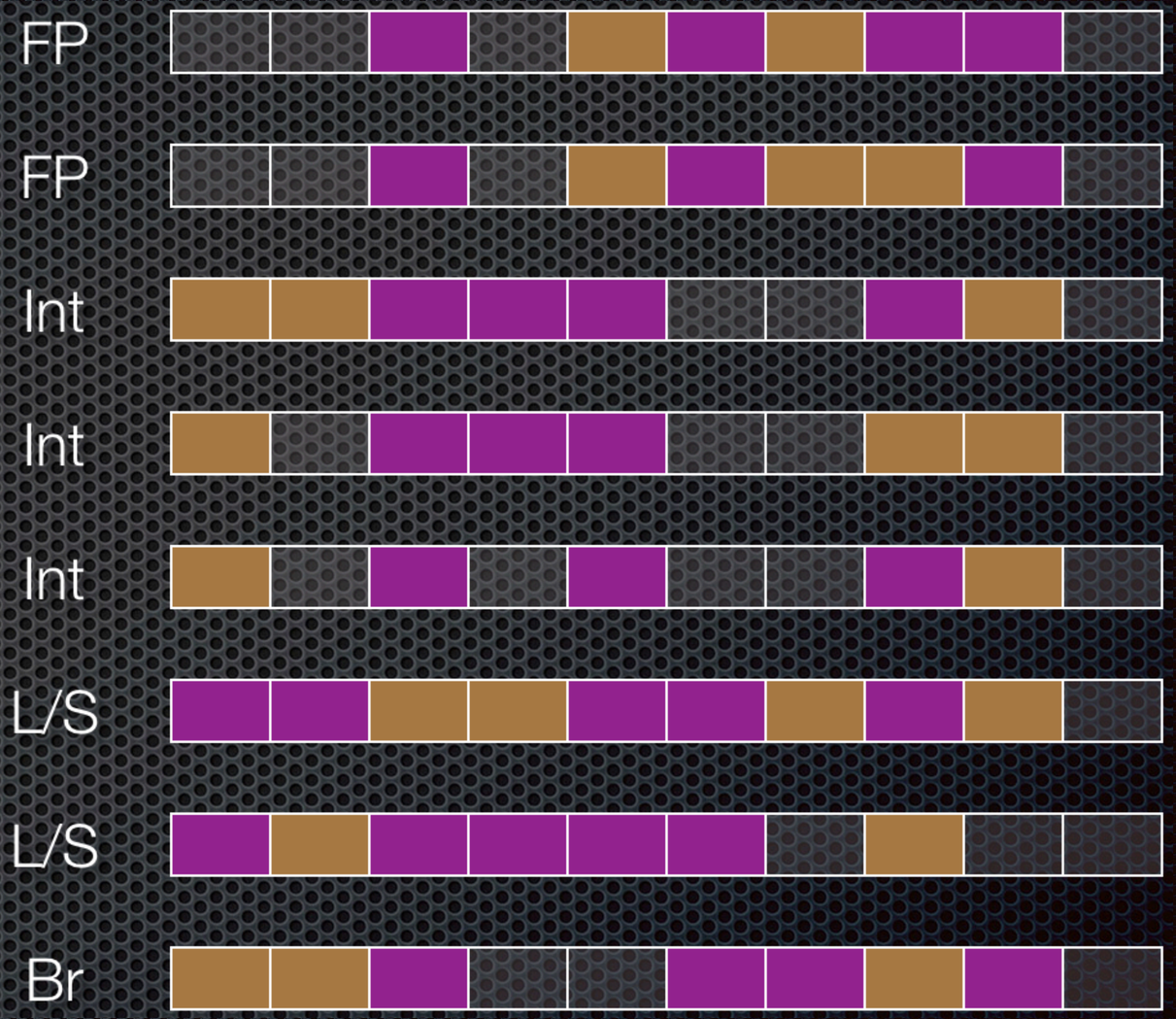
FP	FP	Br	L/S	L/S	Int	Int	L/S	L/S	L/S

FP	FP	L/S	L/S	L/S	Int	Int	Int	Int	Br



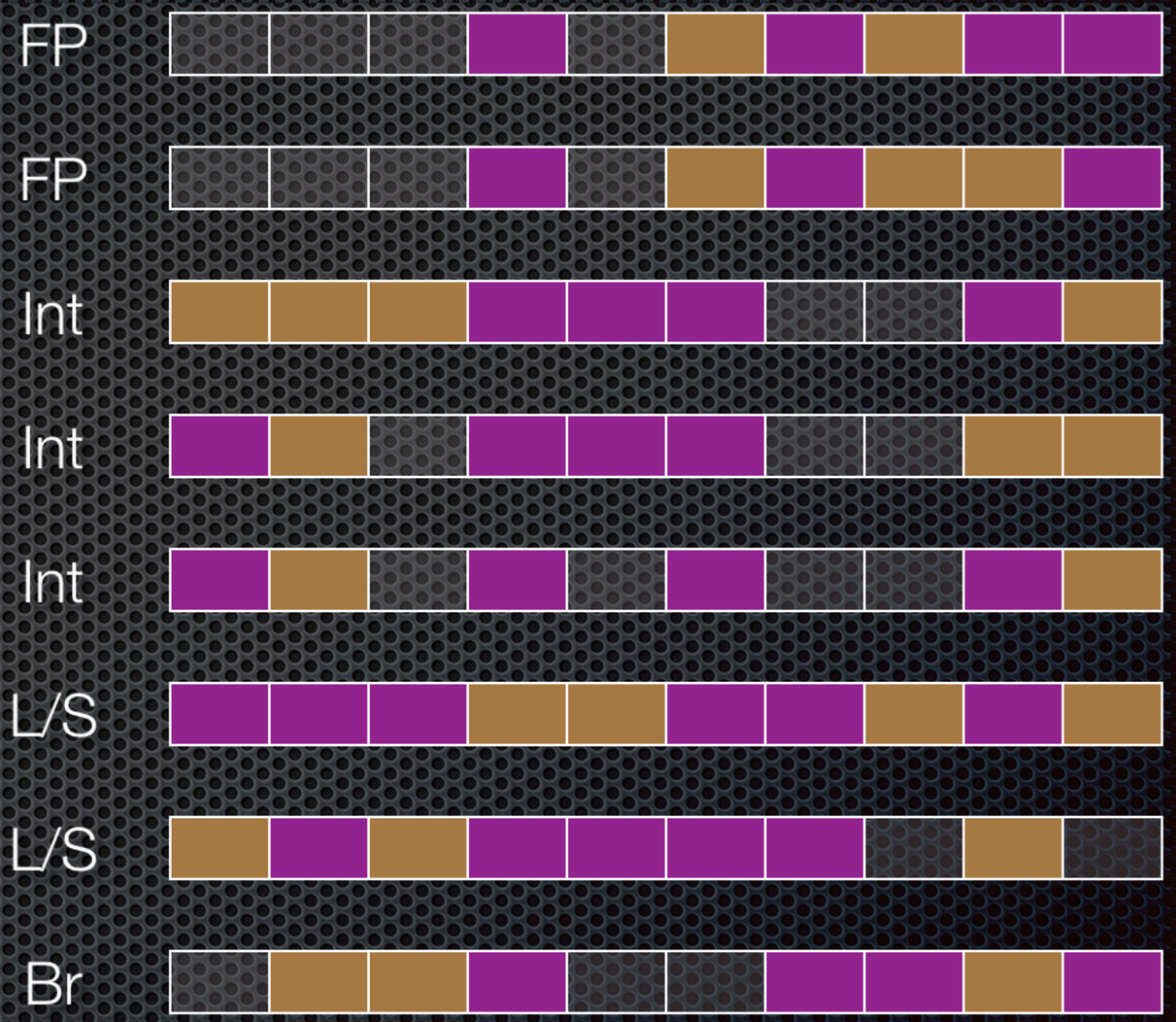
Int	Int	FP	FP	Br	L/S	L/S	Int	Int	L/S

FP	FP	Br	FP	FP	FP	L/S	L/S	L/S	Int



L/S	Int	L/S	Int	Int	FP	FP	Br	L/S	L/S

Br	FP	FP	FP	Br	FP	FP	FP	L/S	L/S



58 slots filled/80 available = 72.5% utilization

Notes

- Original thread is still not done
- Higher throughput, lower per-thread rate
- More pressure on all units
- Memory busy on most cycles
- Branch predictor complications
- Exception handling more complex

Tullsen ISCA 1995

Simultaneous Multithreading: Maximizing On-Chip Parallelism

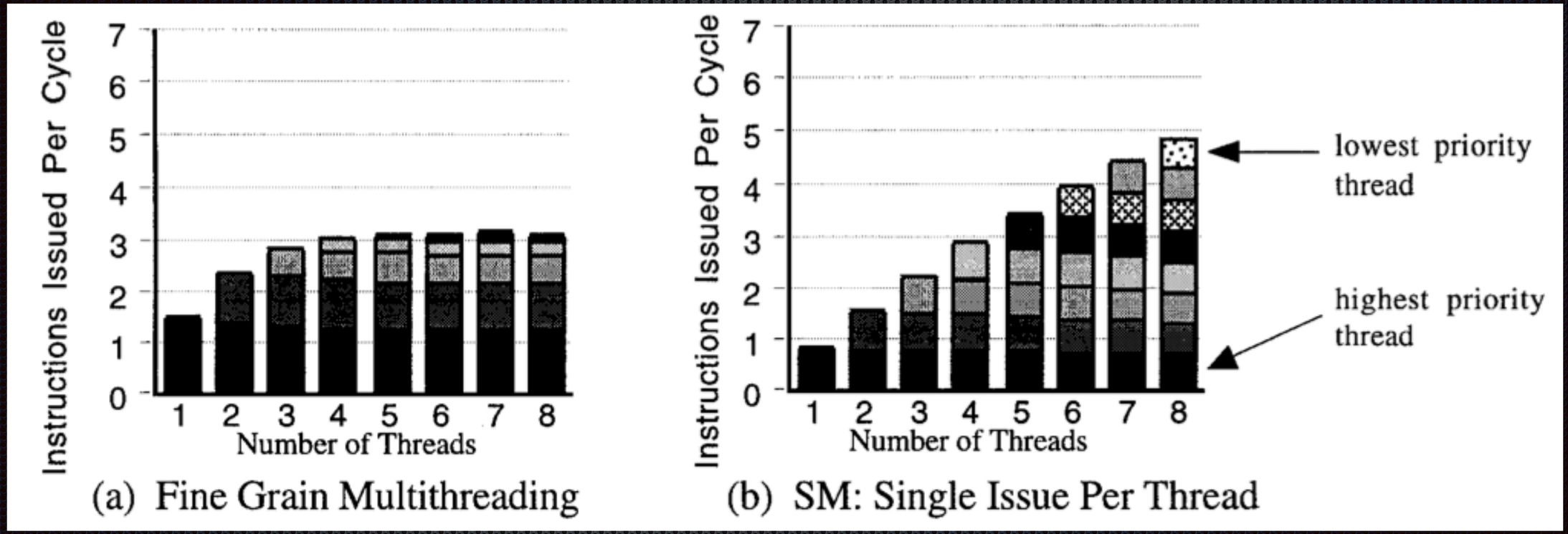
SMT Introduced

- Multithreading originated at least in early CDC machines (early 60s) for I/O latency hiding
- Exploited in fine-grained way for computation in HEP
- Further developed in Tera
- SMT adds ability to go beyond switching threads on each cycle, and blend thread on each cycle

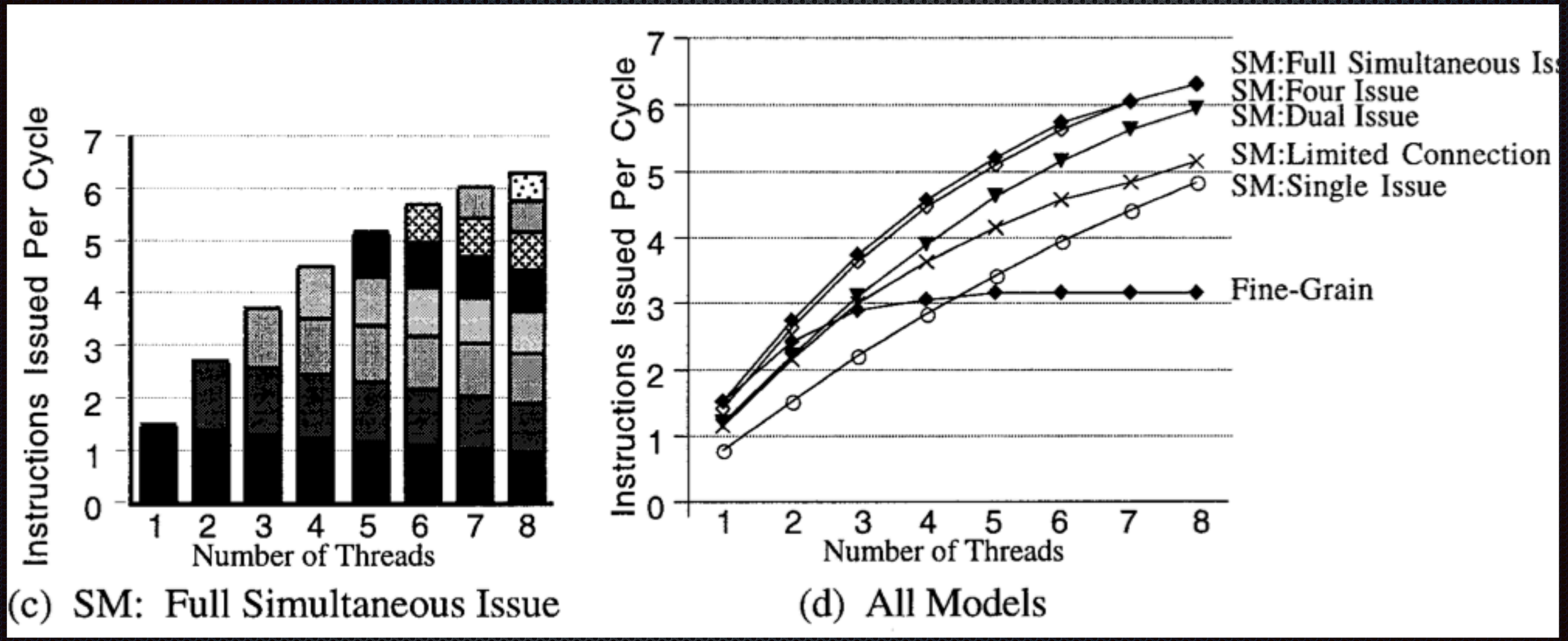
Models

- Single issue per thread per cycle
- Dual, quad, full issue per thread per cycle
- Limited range of connections between thread pipes and functional units
- Compared with fine-grained multithreading

Results


Instructions Issued Per Cycle
0
1
2
3
4
5
6
7
1 2 3 4 5 6 7 8
Number of Threads
(a) Fine Grain Multithreading
Instructions Issued Per Cycle
0
1
2
3
4
5
6
7
1 2 3 4 5 6 7 8
Number of Threads
lowest priority thread
highest priority thread
(b) SM: Single Issue Per Thread

Results


Instructions Issued Per Cycle
0
1
2
3
4
5
6
7
1 2 3 4 5 6 7 8
Number of Threads
(c) SM: Full Simultaneous Issue
Instructions Issued Per Cycle
SM:Full Simultaneous Iss
SM:Four Issue
SM:Dual Issue
SM:Limited Connection
SM:Single Issue
Fine-Grain
Number of Threads
(d) All Models

Compared to Multicore

Purpose of Test	Common Elements	Specific Configuration	Throughput (instructions/cycle)
Unlimited FUs: equal total issue bandwidth, equal number of register sets (processors or threads)	**Test A**: FUs = 32 Issue bw = 8 Reg sets = 8	SM: 8 thread, 8-issue	6.64
		MP: 8 1-issue procs	5.13
	Test B: FUs = 16 Issue bw = 4 Reg sets = 4	SM: 4 thread, 4-issue	3.40
		MP: 4 1-issue procs	2.77
	Test C: FUs = 16 Issue bw = 8 Reg sets = 4	SM: 4 thread, 8-issue	4.15
		MP: 4 2-issue procs	3.44
Unlimited FUs: Test A, but limit SM to 10 FUs	**Test D**: Issue bw = 8 Reg sets = 8	SM: 8 thread, 8 issue, 10 FU	6.36
		MP: 8 1-issue procs, 32 FU	5.13
Unequal Issue BW: MP has up to four times the total issue bandwidth	**Test E**: FUs = 32 Reg sets = 8	SM: 8 thread, 8-issue	6.64
		MP: 8 4-issue procs	6.35
	Test F: FUs = 16 Reg sets = 4	SM: 4 thread, 8-issue	4.15
		MP: 4 4-issue procs	3.72
FU Utilization: equal FUs, equal issue bw, unequal reg sets	**Test G**: FUs = 8 Issue BW = 8	SM: 8 thread, 8-issue	5.30
		MP: 2 4-issue procs	1.94

Figure 5: **Results for the various multiprocessor vs. simultaneous multithreading comparisons. The multiprocessor always has one functional unit of each type per processor. In most cases the SM processor has the same total number of each FU type as the MP.**

Discussion

Mahlke ISCA 1995

A Comparison of Full and Partial Predicated Execution Support for ILP Processors

Predication

- Added operand that specifies predicate for execution of an instruction
- If satisfied, instruction completes, otherwise squashed
- Full predication adds capability to all instructions
- Partial supports some (e.g., conditional moves)
  - Depends on speculation

Implementation

- Set of predicate registers
- Predicate define instructions
- Instructions refer to defined predicates
- Similar to Itanium
- Different from ARM-32, where each instruction can conditionally set a condition, and be predicated on existing conditions (ARM-64 more limited)

Compilation

- Uses hyperblocks (connected series of basic blocks) with single entry, multiple exit
- Control flow between blocks eliminated by if-conversion
- Create fully predicated code, then transform to partially predicated code
  - Significant code expansion (roughly double)
- Then peephole optimize

Simple 8-issue Model

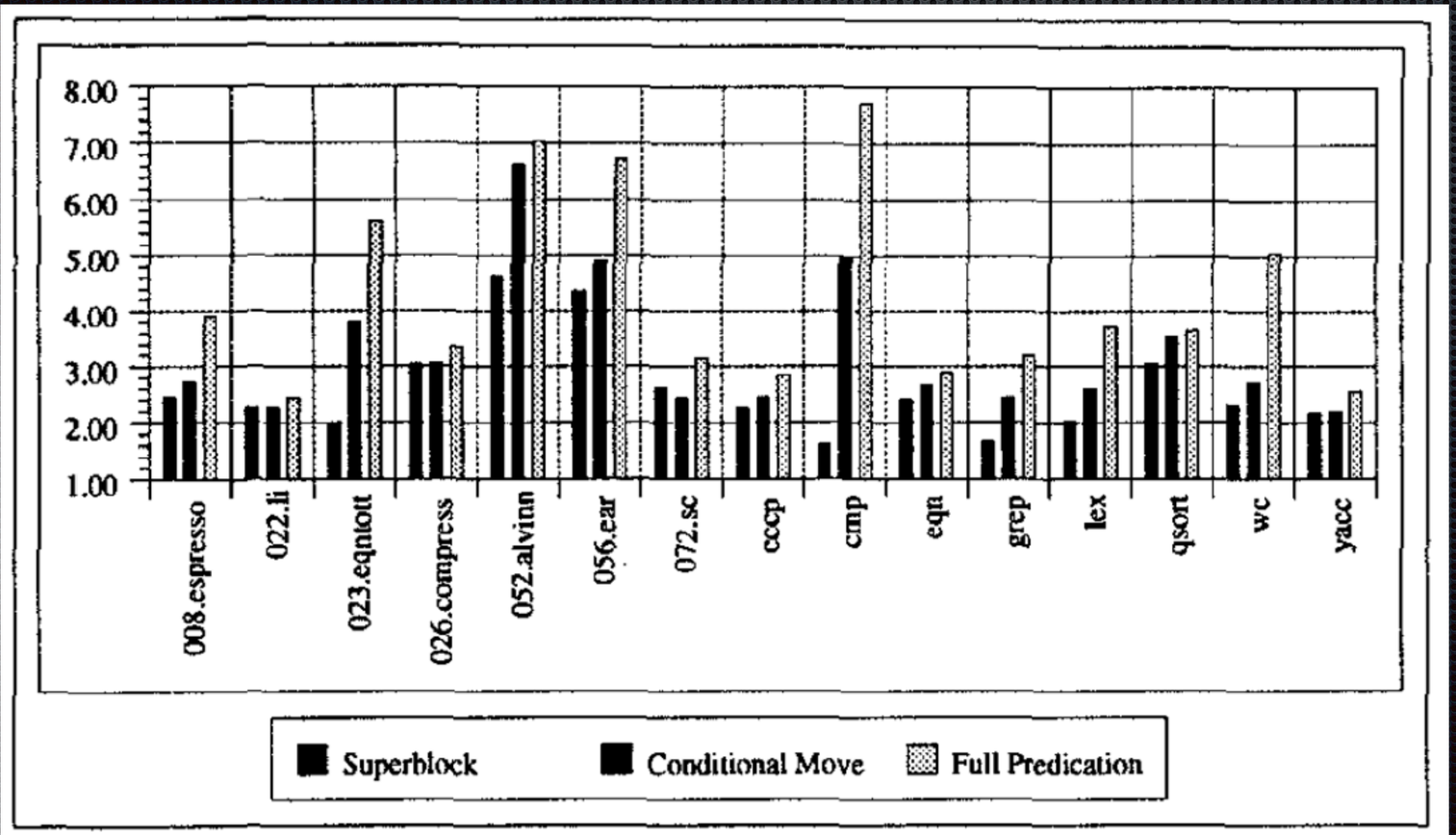


Figure 8: Effectiveness of full and partial predicate support for an 8-issue, 1-branch processor with perfect caches.

2-Branch Speculation

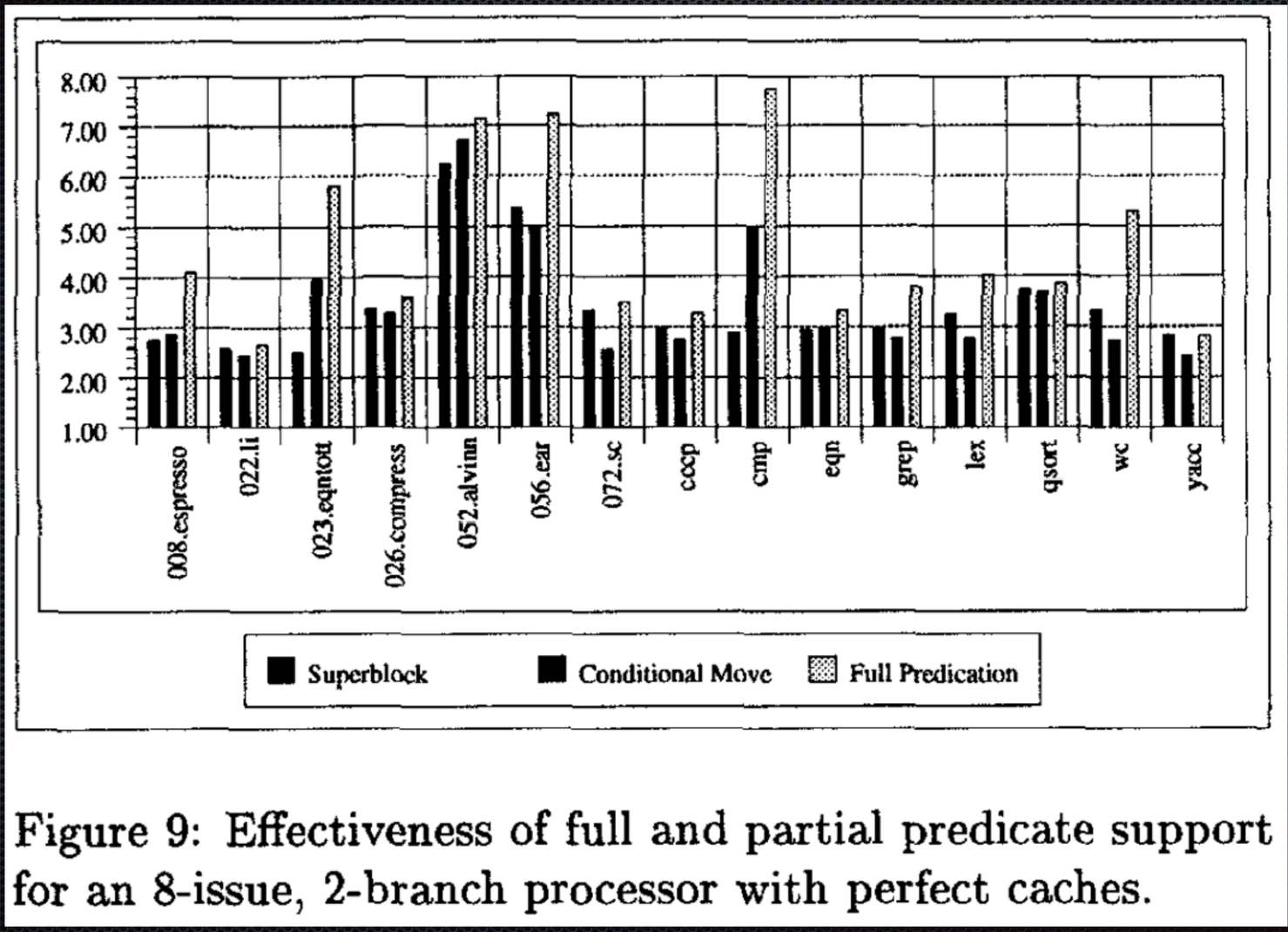


Figure 9: Effectiveness of full and partial predicate support for an 8-issue, 2-branch processor with perfect caches.

Simple 4-issue Model

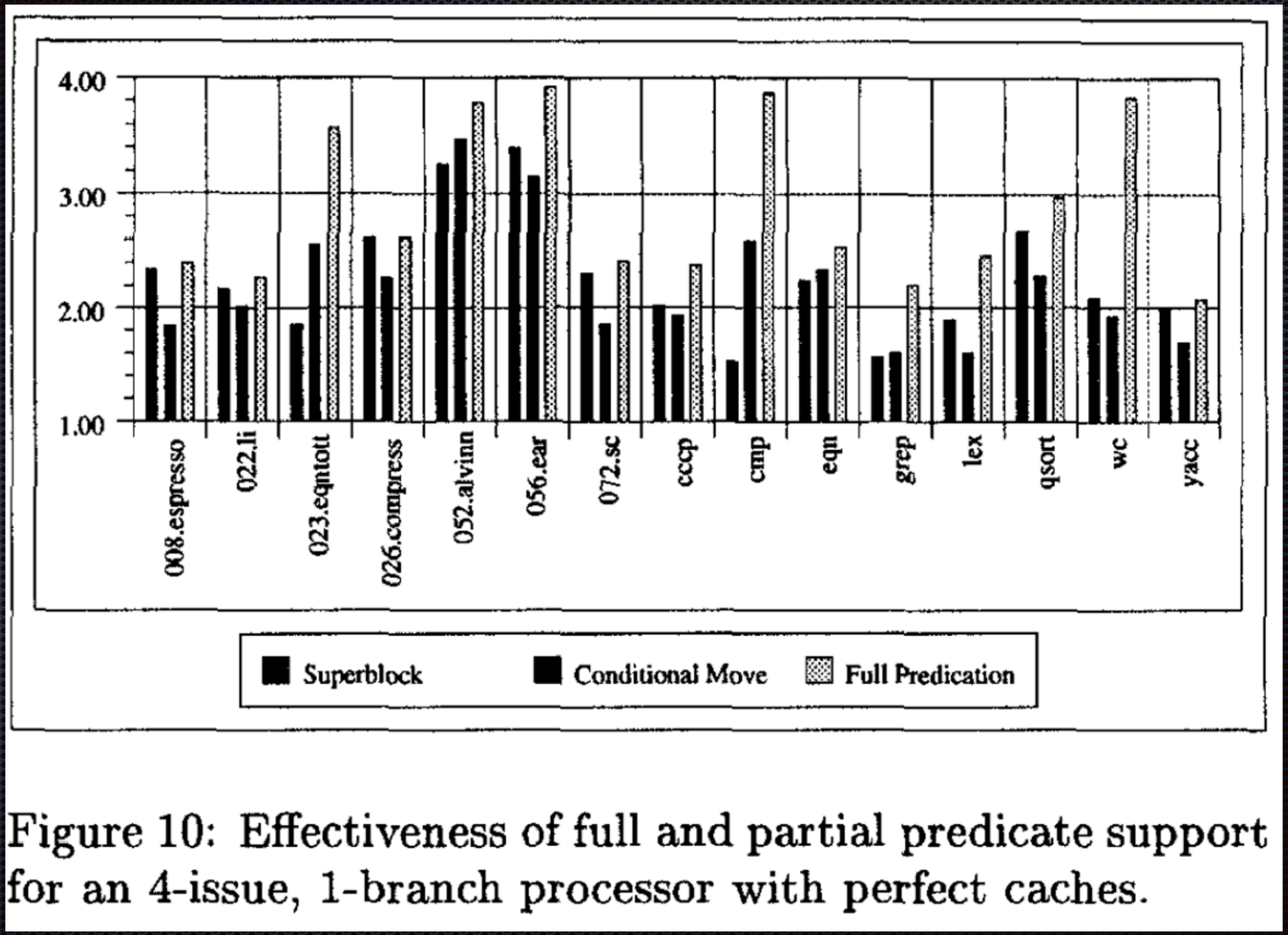


Figure 10: Effectiveness of full and partial predicate support for an 4-issue, 1-branch processor with perfect caches.

Realistic Caches

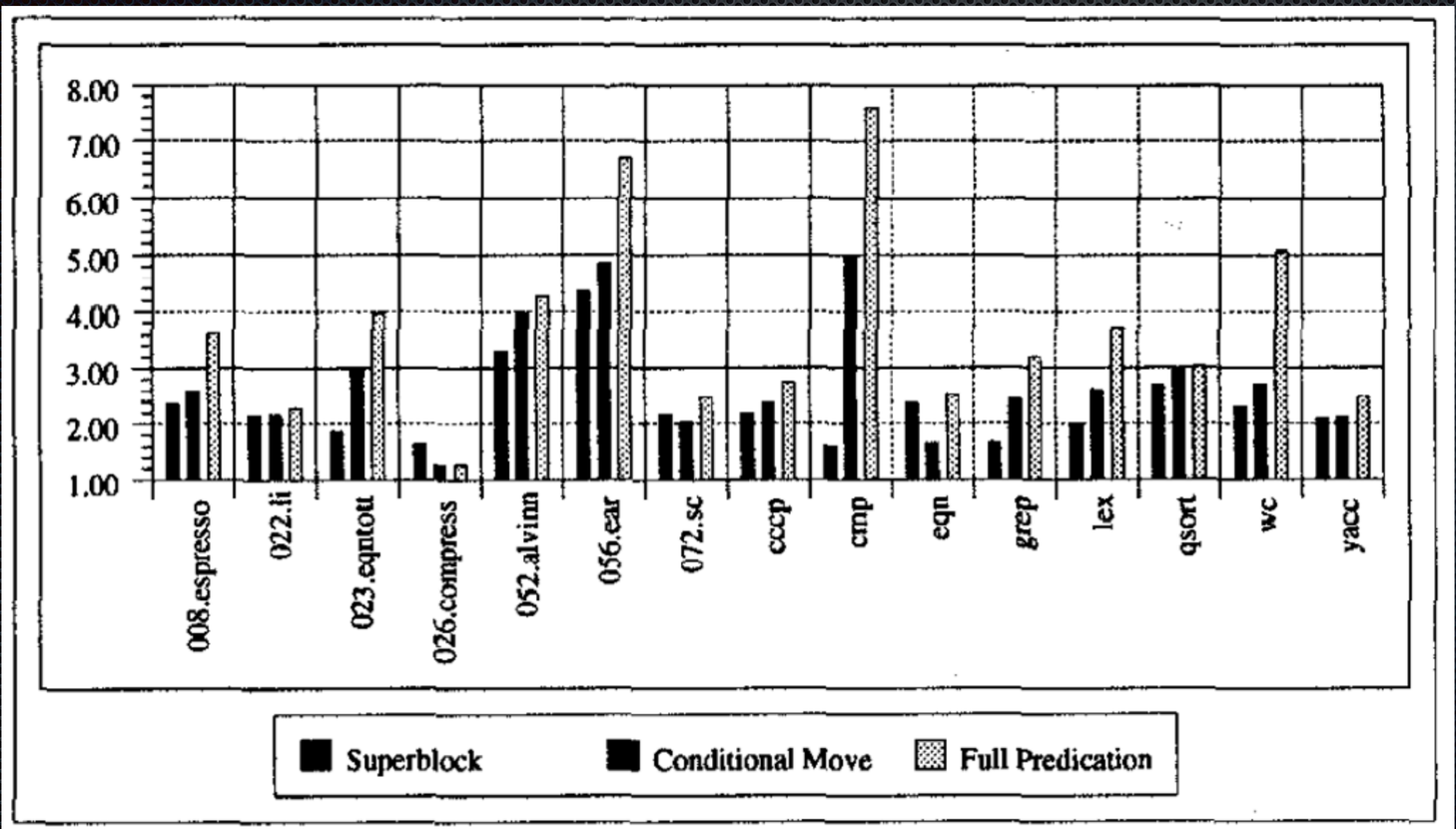


Figure 11: Effectiveness of full and partial predicate support for an 8-issue, 1-branch processor with 64K instruction and data caches.

Effect on Instruction Count

Benchmark	Superblk	Cond. Move	Full Pred.
008.espresso	489M	812M (1.66)	626M (1.28)
022.li	31M	38M (1.23)	32M (1.04)
023.eqntott	1030M	1230M (1.19)	885M (0.86)
026.compress	90M	128M (1.41)	108M (1.20)
052.alvinn	3574M	4003M (1.12)	3603M (1.01)
056.ear	11225M	13838M (1.23)	11073M (0.99)
072.sc	91M	85M (0.93)	75M (0.83)
cccp	3701K	5077K (1.37)	3855K (1.04)
cmp	932K	1422K (1.53)	922K (0.99)
eqn	44M	49M (1.11)	44M (0.99)
grep	1282K	2467K (1.92)	1647K (1.28)
lex	36M	75M (2.10)	46M (1.29)
qsort	44M	70M (1.61)	49M (1.11)
wc	1493K	2999K (2.01)	1526K (1.02)
yacc	43M	66M (1.53)	50M (1.16)

Table 2: Dynamic instruction count comparison.

Effect on Rest of Branches

Benchmark	Superblock			Conditional Move			Full Predication		
	BR	MP	MPR	BR	MP	MPR	BR	MP	MPR
008.espresso	75M	3402K	4.55%	38M	2066K	5.38%	33M	1039K	3.15%
022.li	7457K	774K	10.38%	6169K	694K	11.25%	6110K	702K	11.5%
023.eqntott	315M	42M	13.47%	53M	6732K	12.66%	51M	6931K	13.57%
026.compress	12M	1344K	10.9%	9269K	864K	9.32%	9240K	867K	9.38%
052.alvinn	463M	1091K	0.24%	74M	896K	1.23%	74M	1032K	1.38%
056.ear	1539M	66M	4.3%	443M	16M	3.52%	442M	15M	3.4%
072.sc	22M	1232K	5.49%	11M	1044K	9.19%	11M	934K	8.26%
cccp	921K	66K	7.19%	537K	65K	12.17%	534K	65K	12.15%
cmp	530K	4395	0.83%	26K	31	0.12%	26K	31	0.12%
eqn	7470K	1612K	8.2%	4506K	514K	11.4%	4495K	511K	11.37%
grep	663K	9660	1.46%	171K	20K	11.7%	171K	20K	11.73%
lex	14M	232K	1.65%	3070K	201K	6.55%	3030K	196K	6.46%
qsort	8847K	1332K	15.06%	6092K	597K	9.79%	6066K	610K	10.06%
wc	478K	33K	6.85%	224K	57	.025%	224K	57	.025%
yacc	12M	517K	4.31%	5944K	445K	7.48%	5900K	431K	7.31%

BRanches, MisPredicts, MisPredict Rate

Discussion