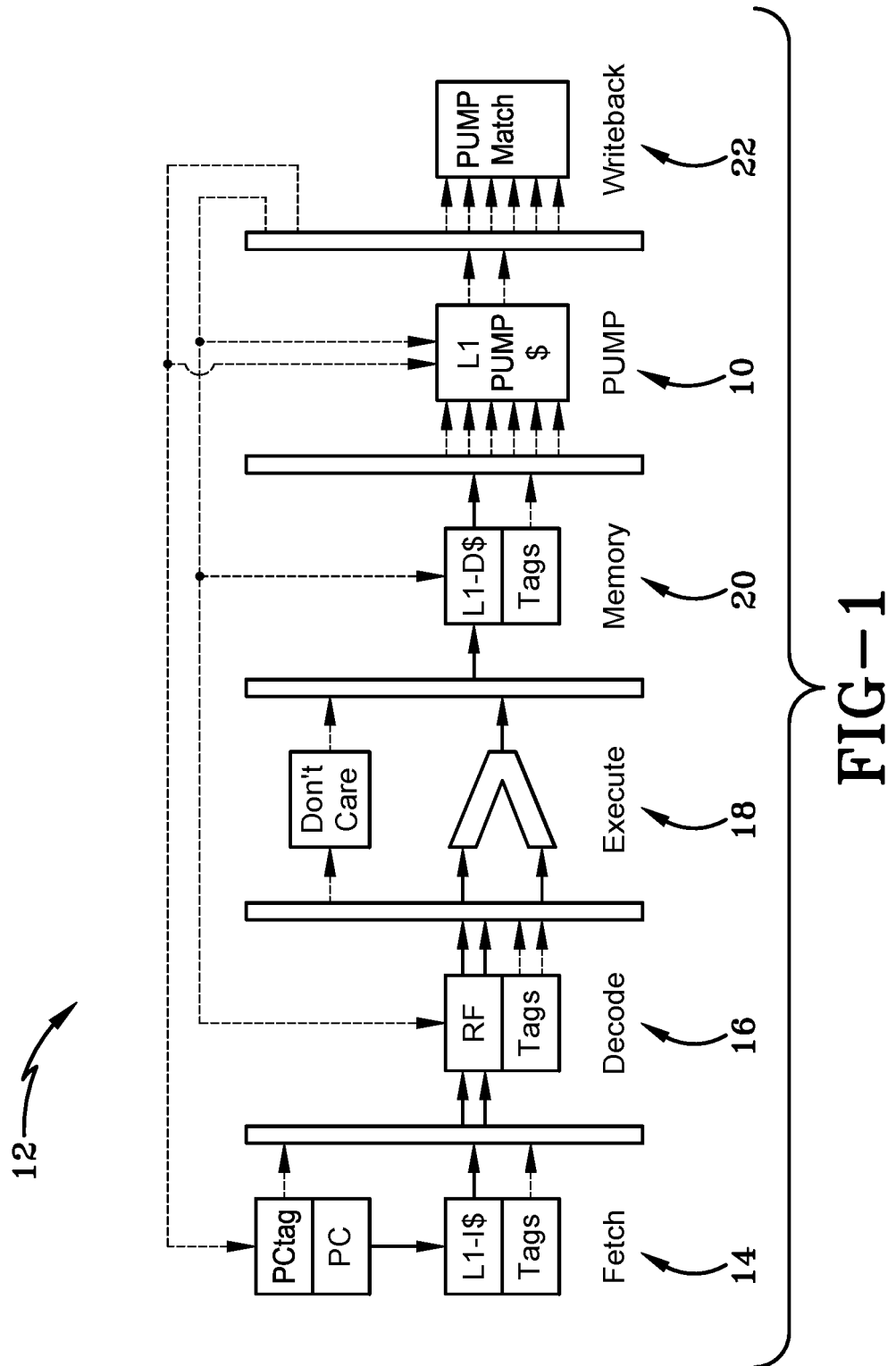
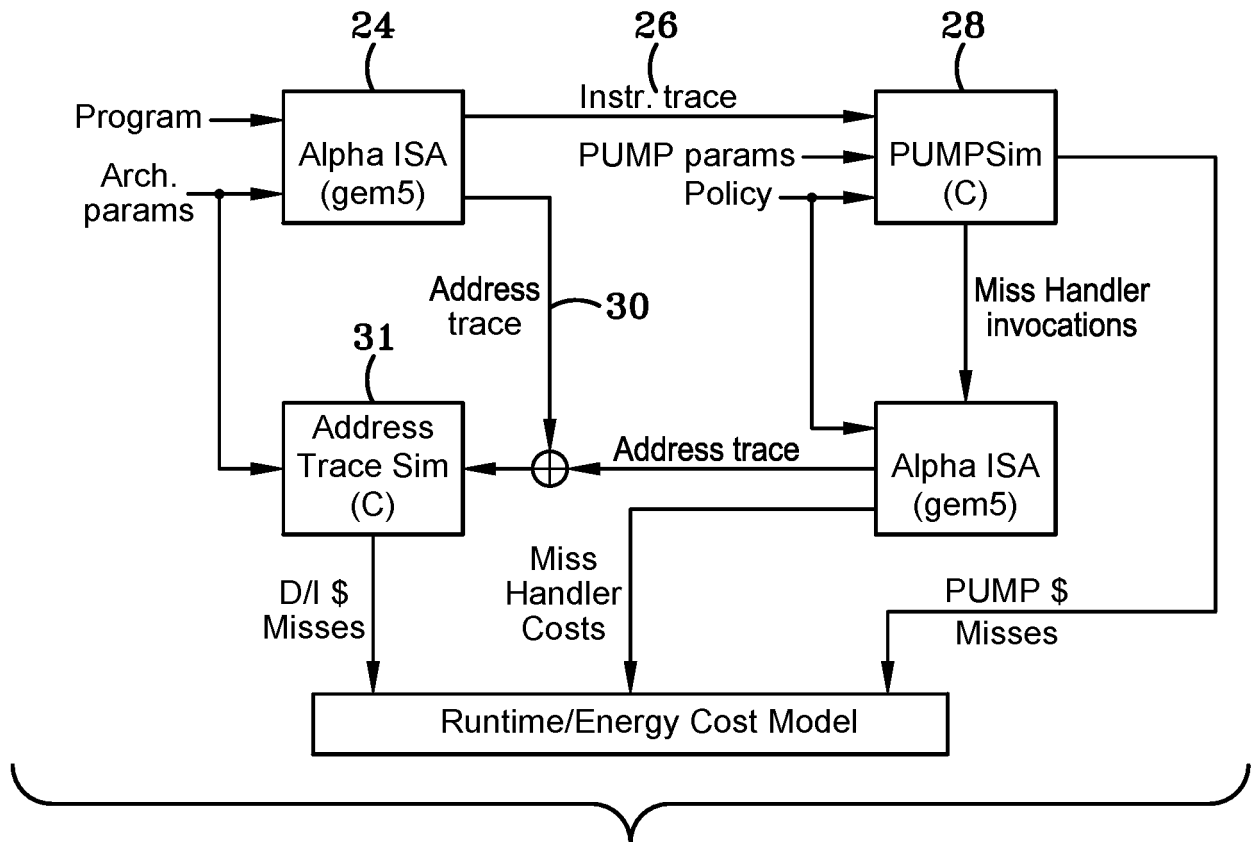
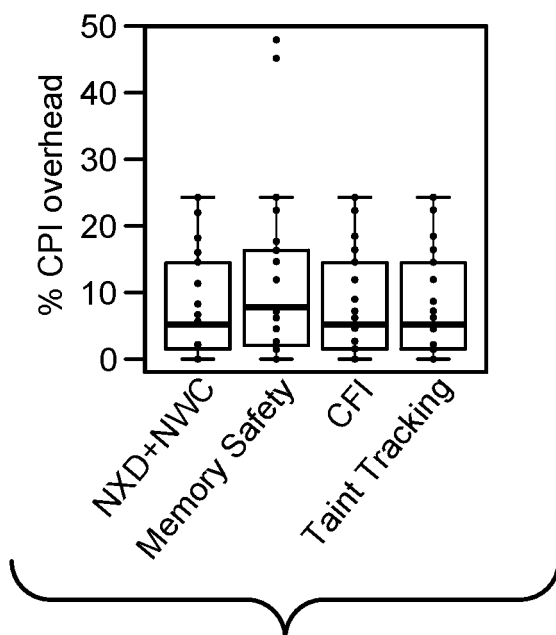
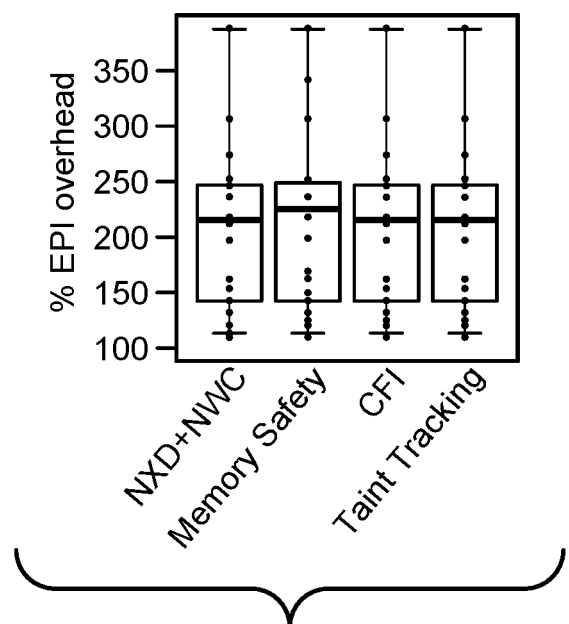




**FIG-2****FIG-3A****FIG-3B**

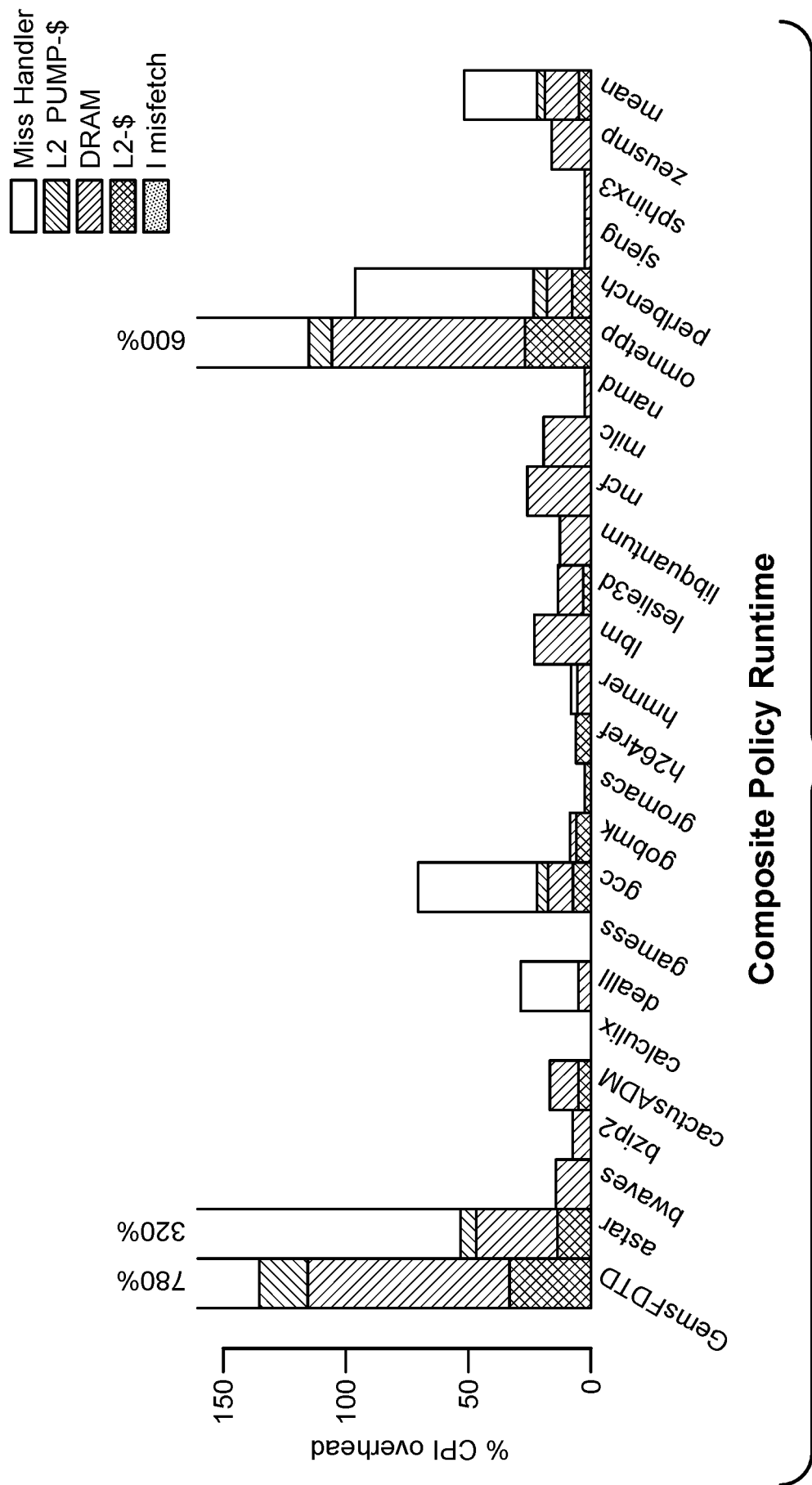


FIG-4A

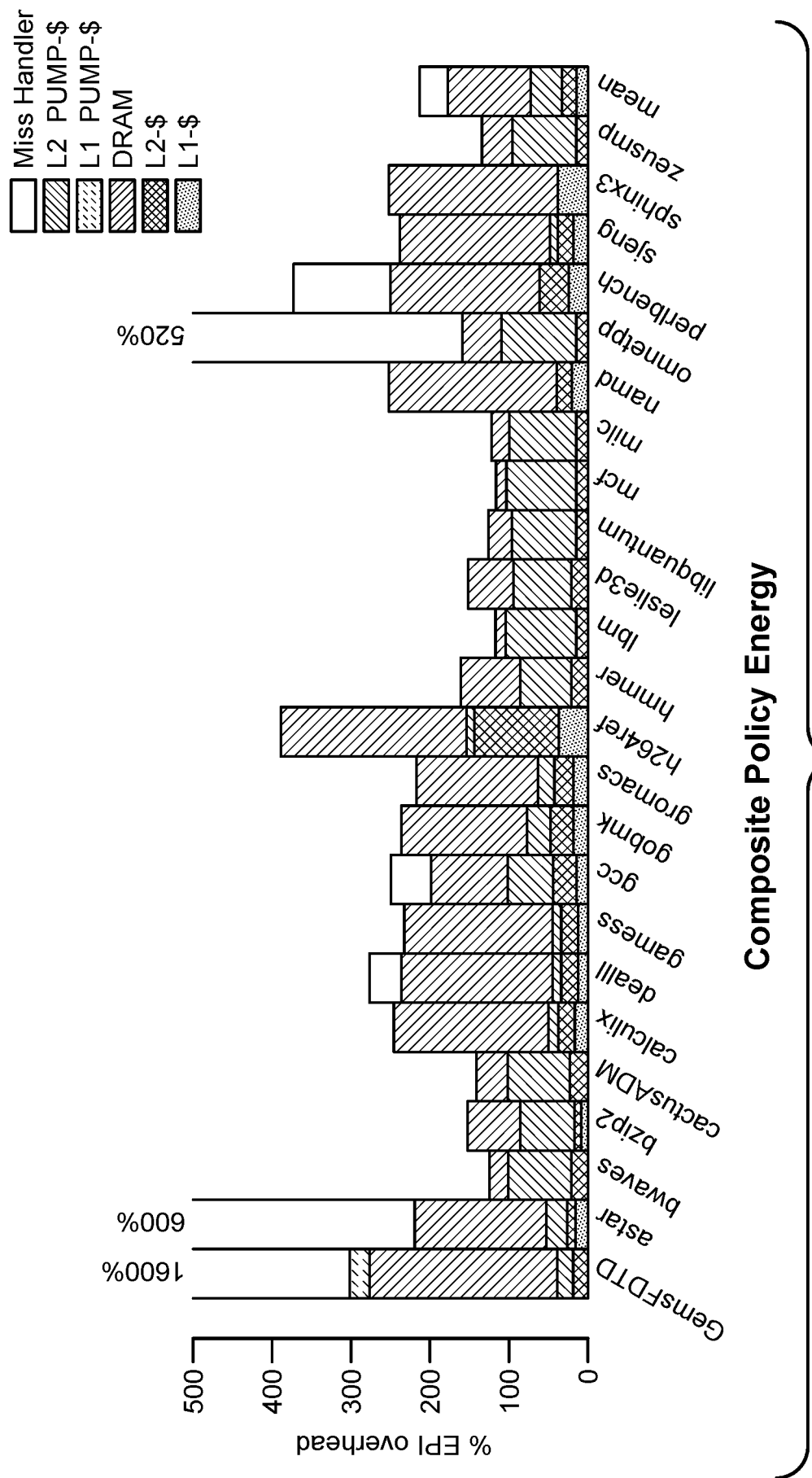
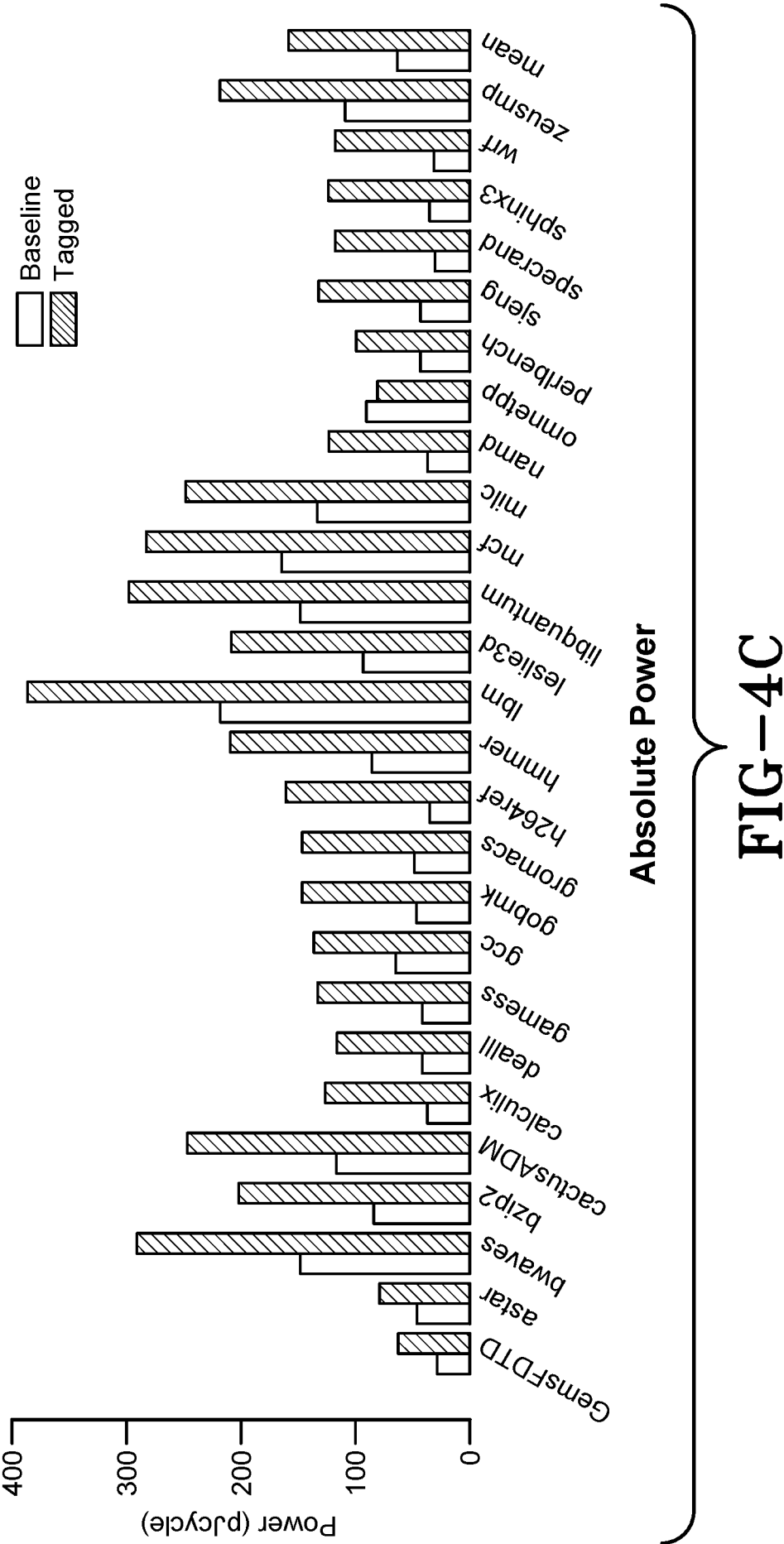
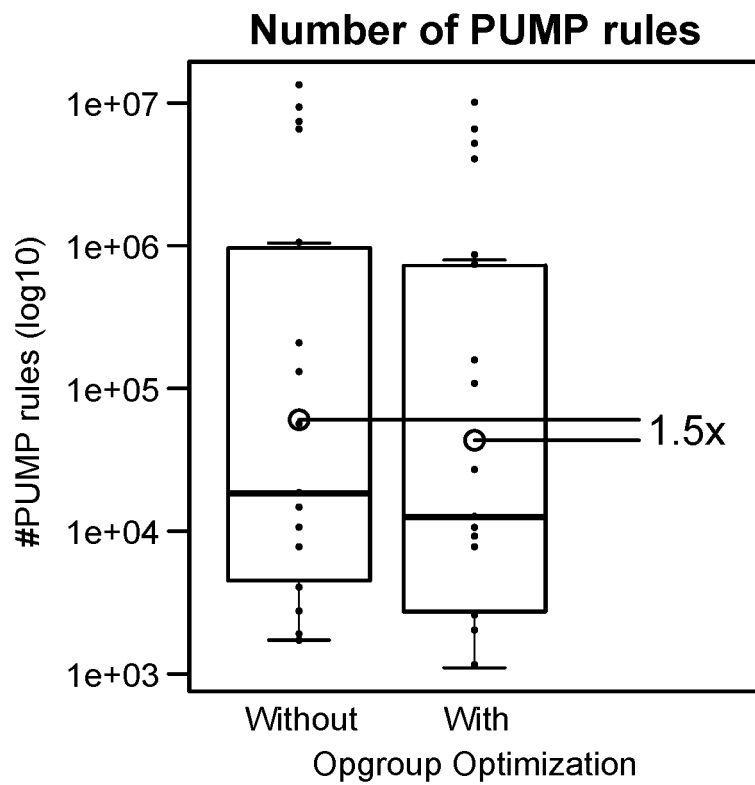
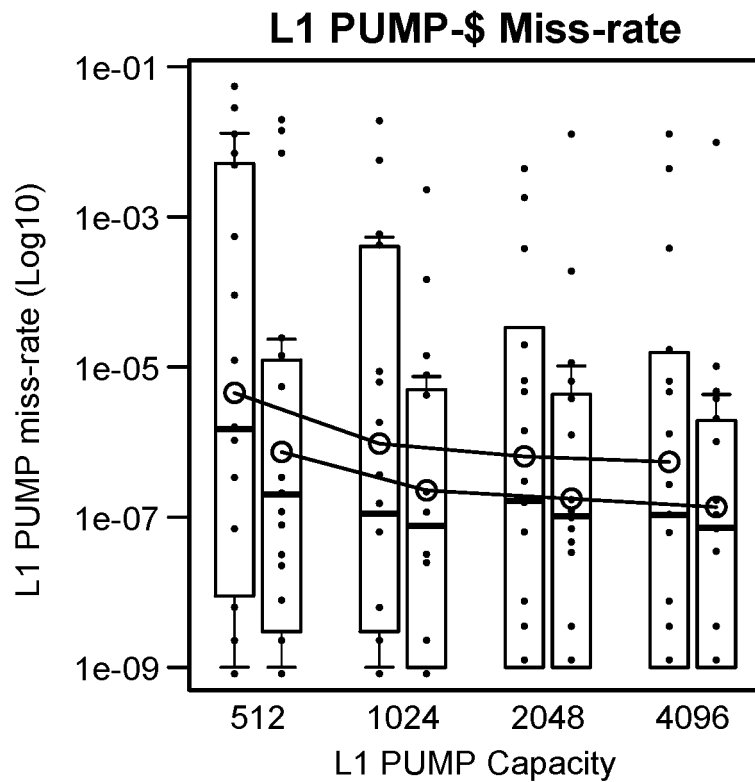


FIG-4B



**FIG-5A****FIG-5B**

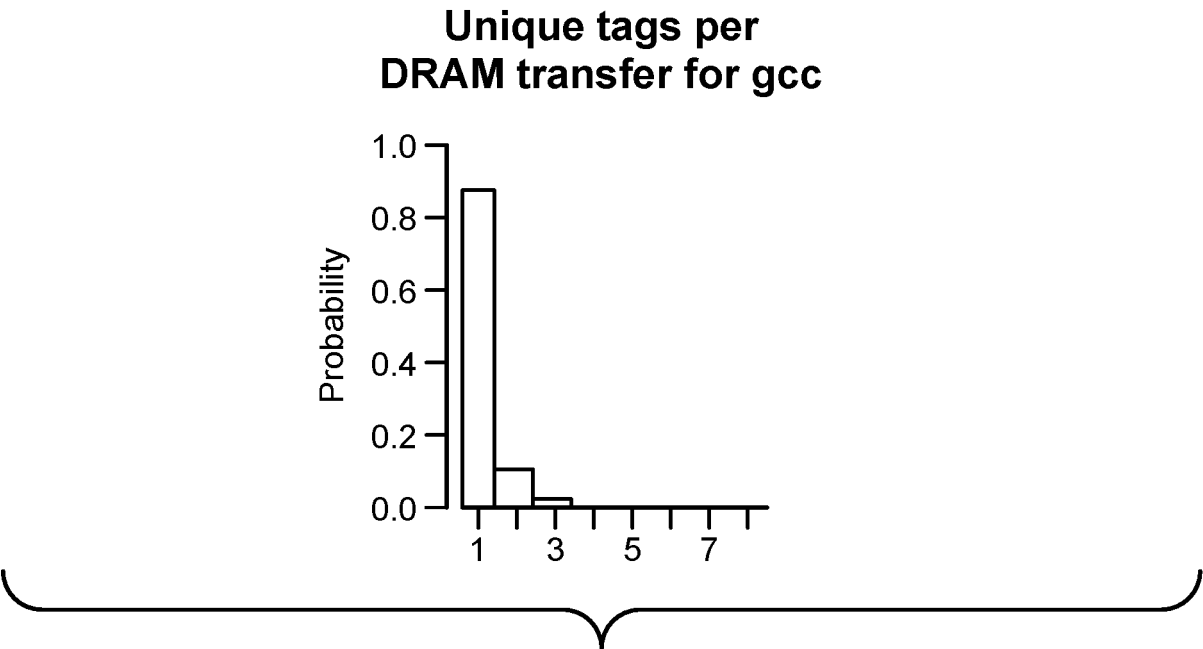


FIG-6A

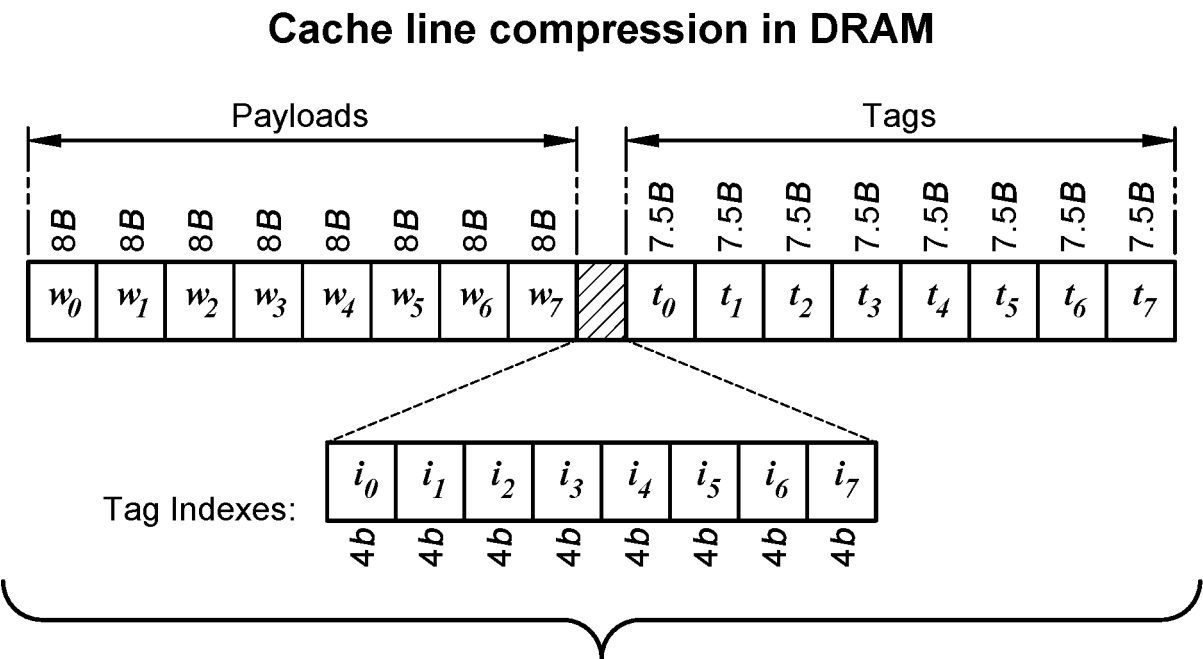
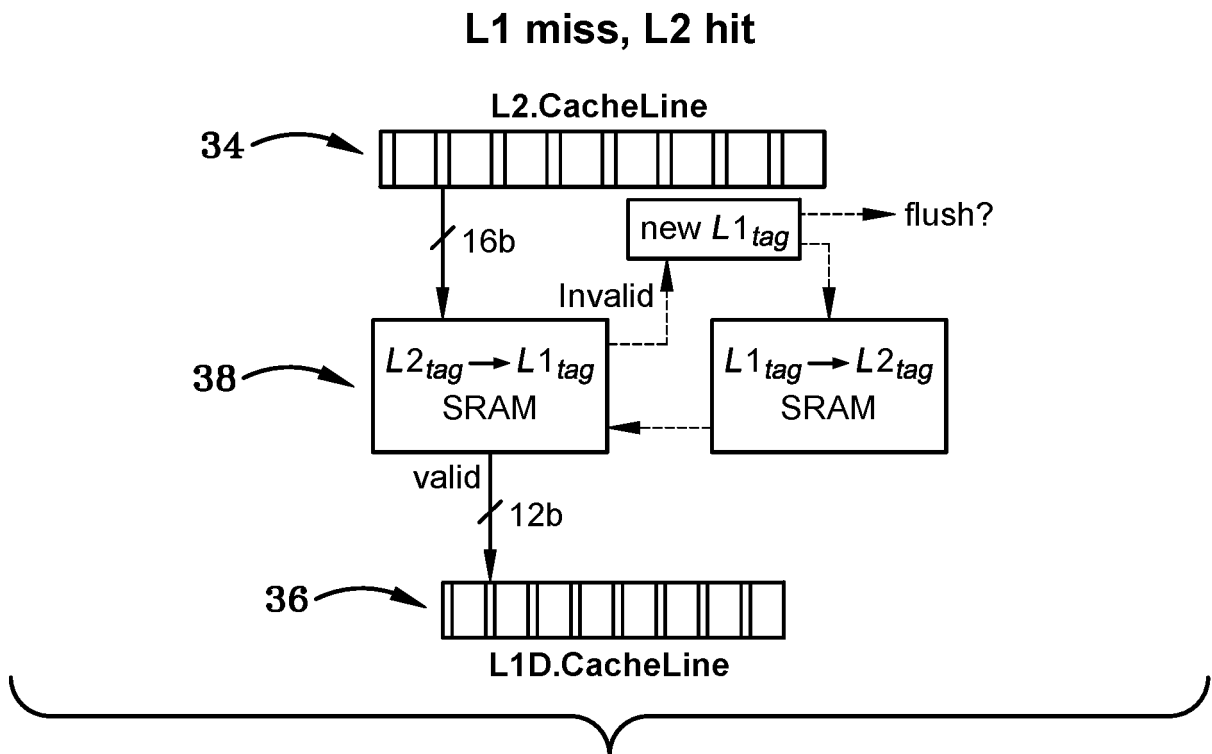
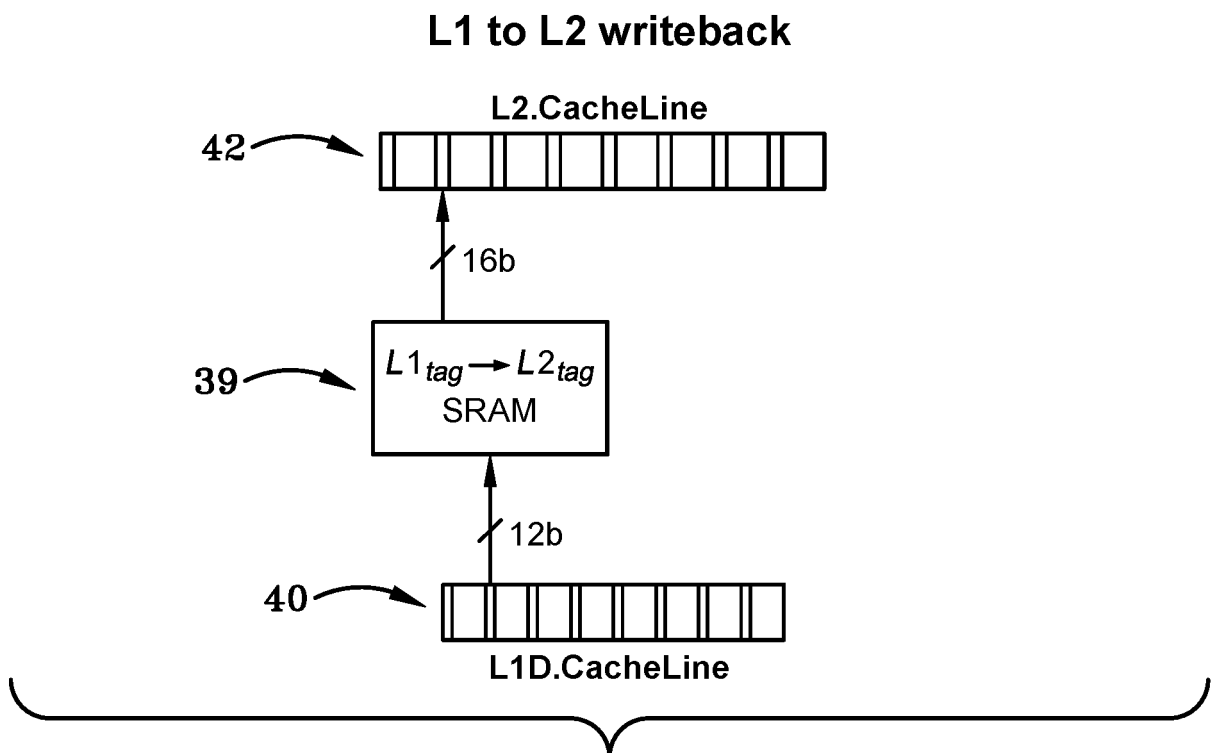
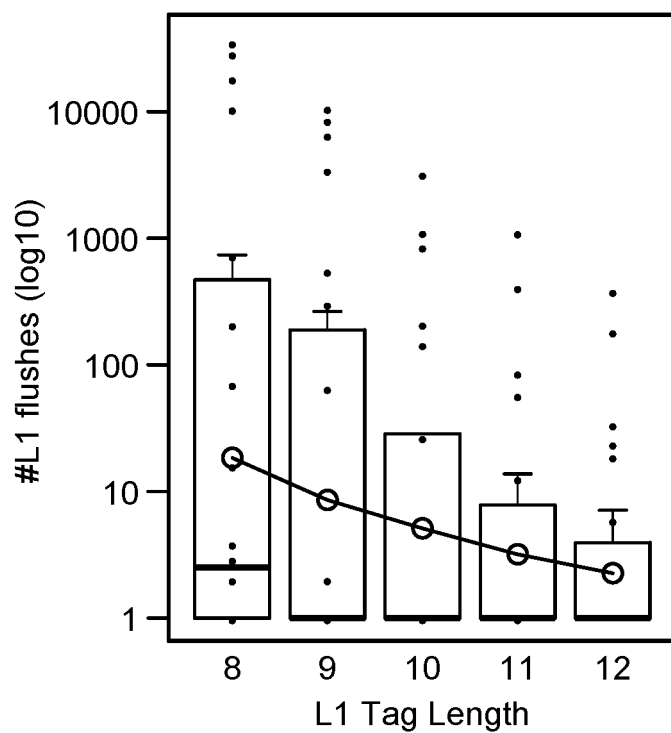
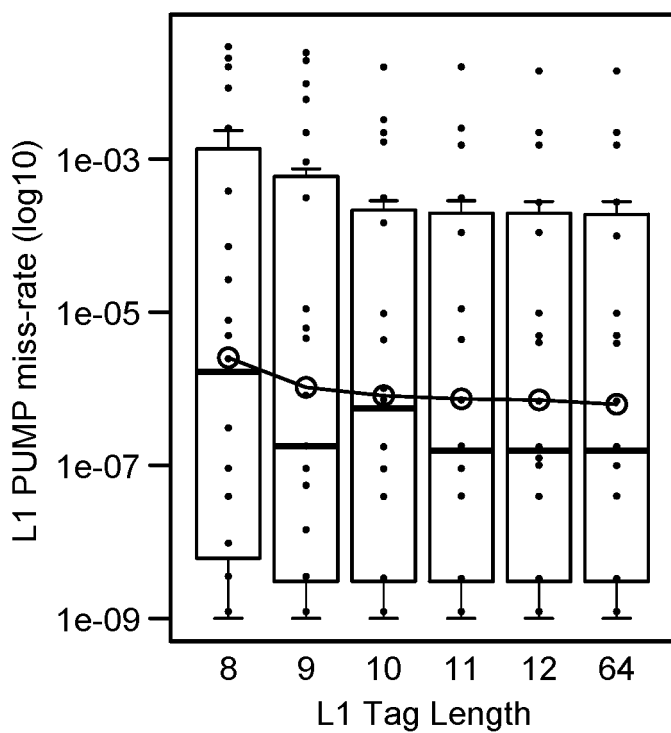
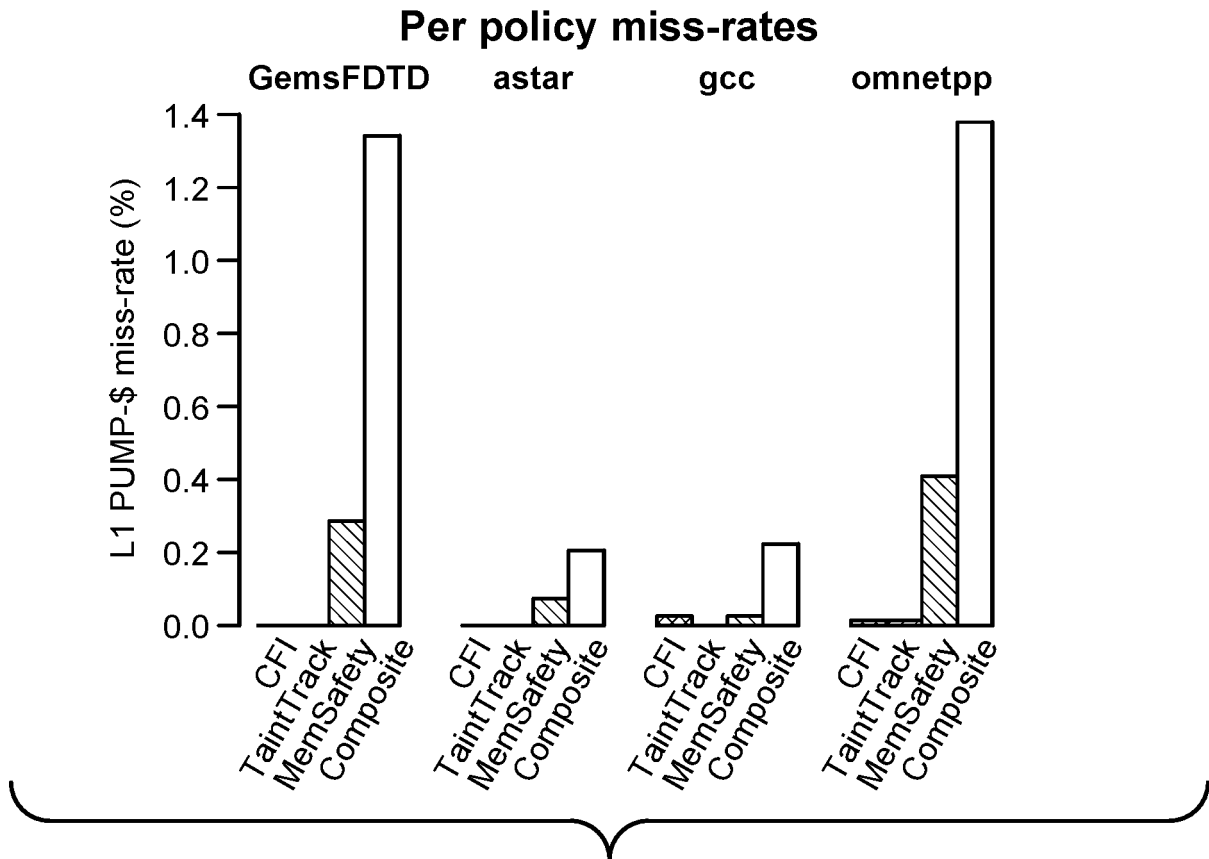
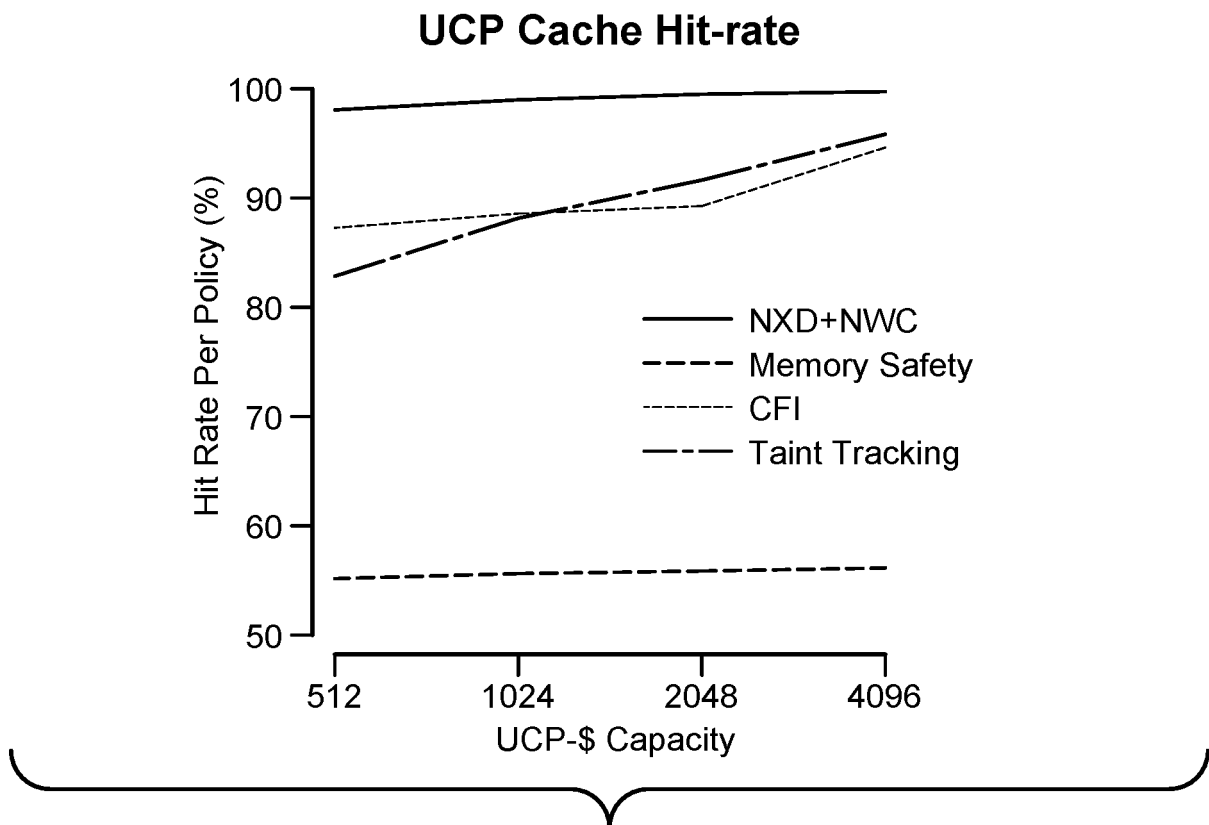


FIG-6B

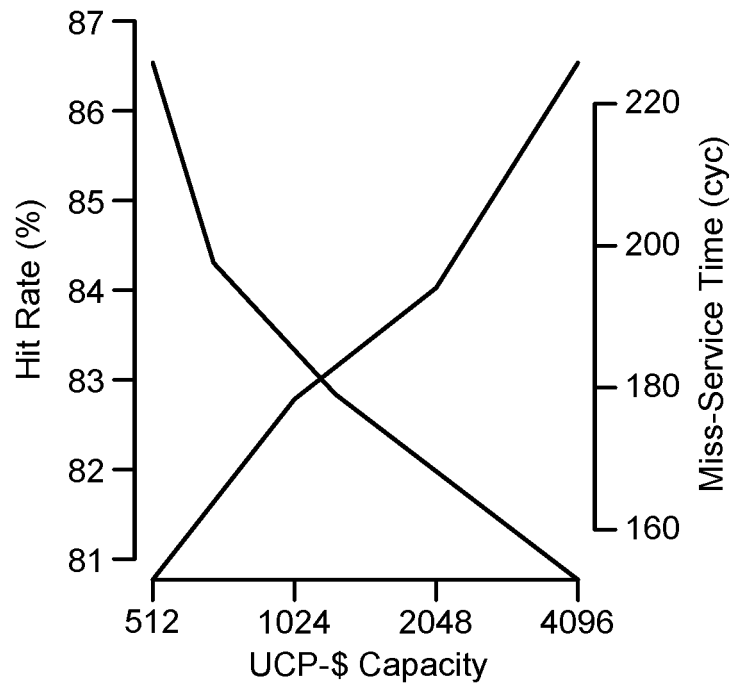
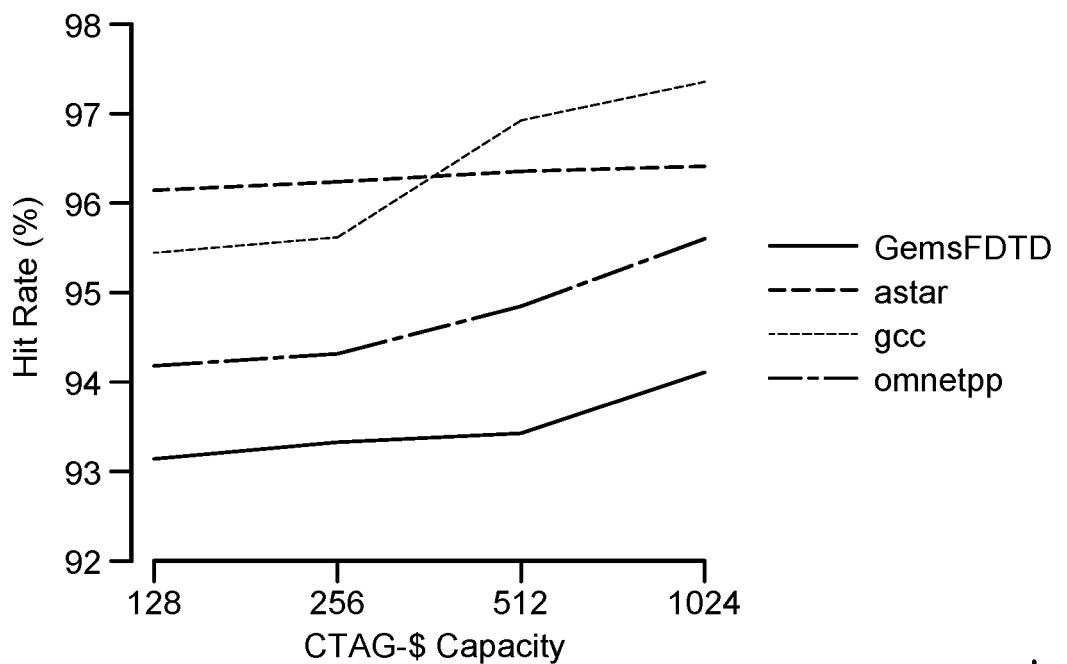
**FIG-7A****FIG-7B**

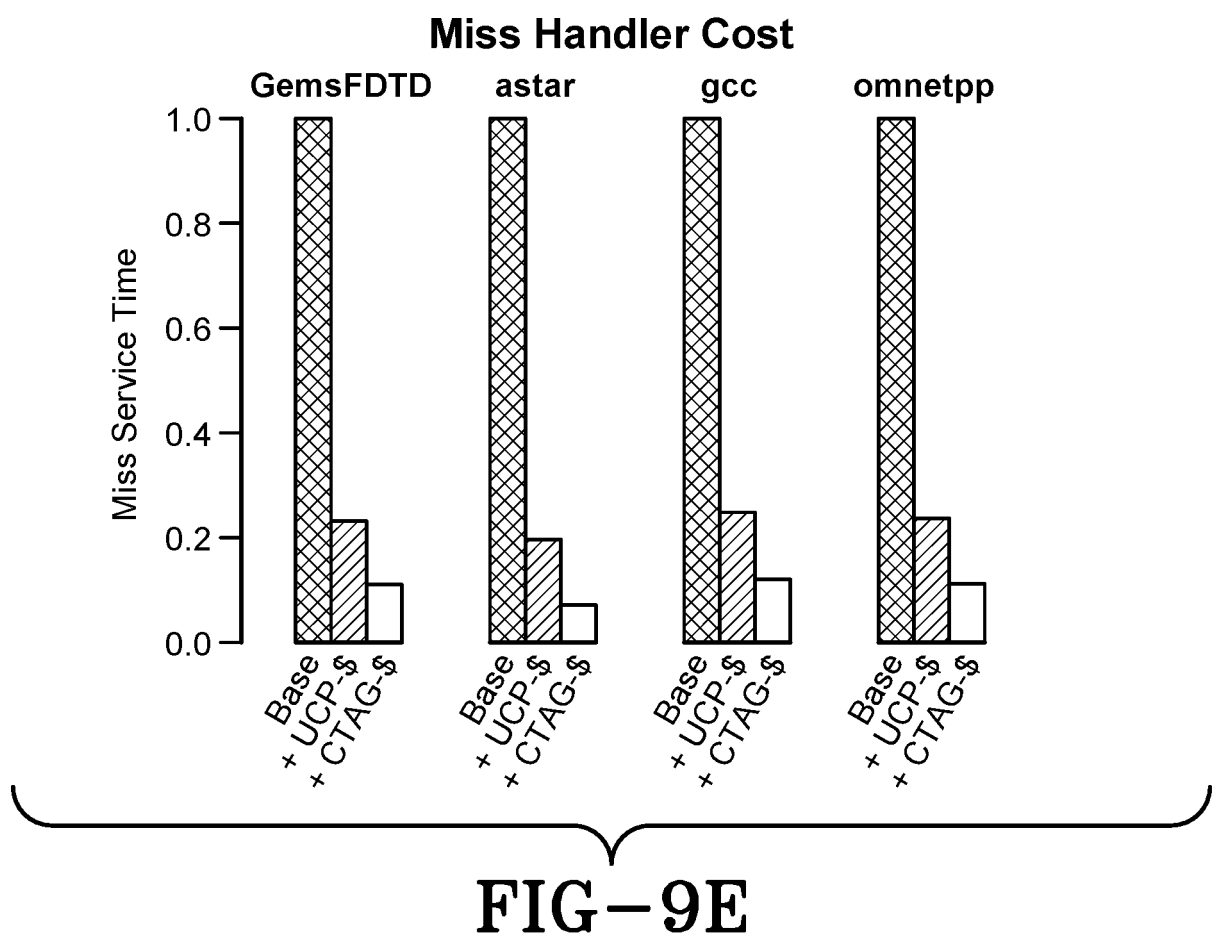
**FIG-8A****FIG-8B**

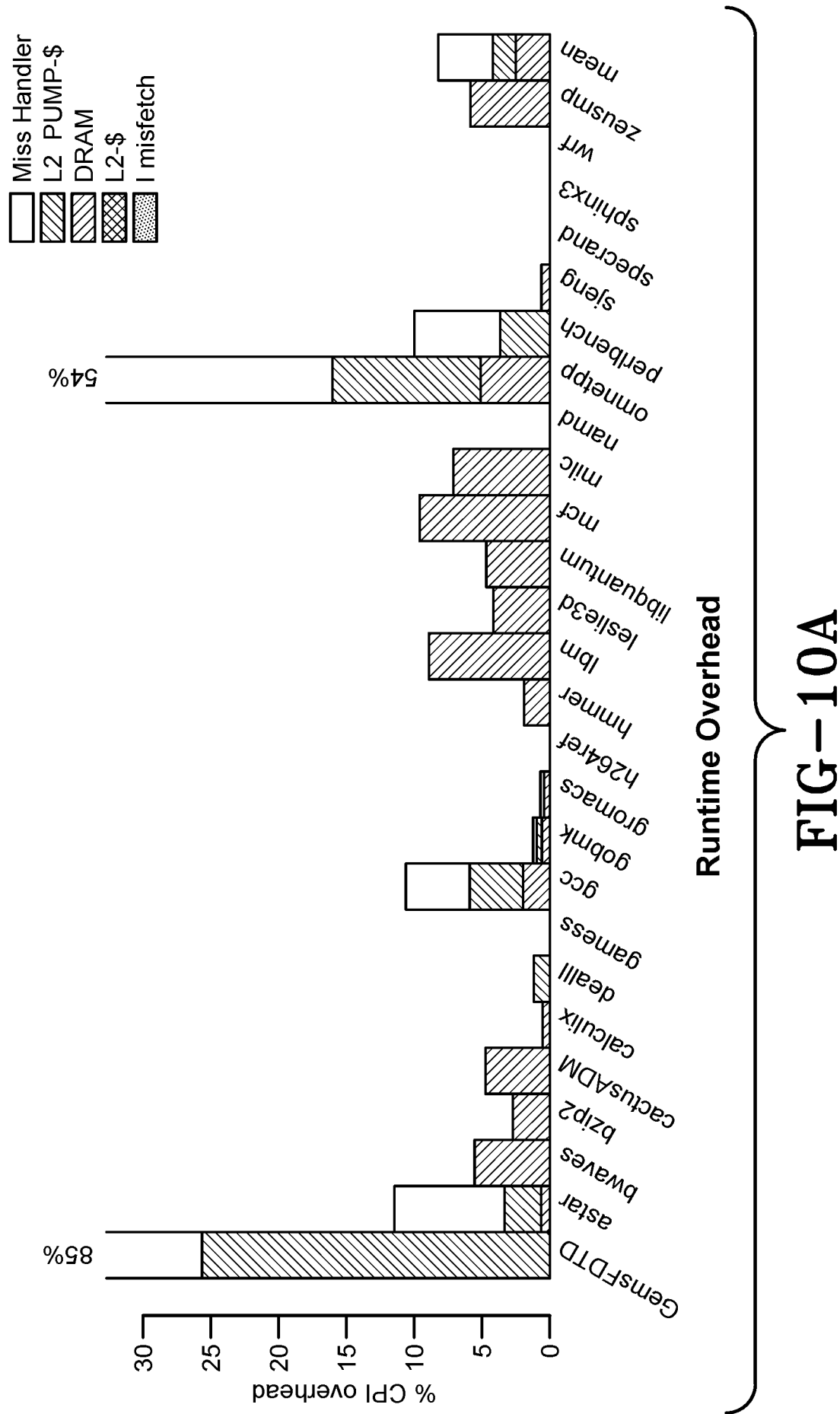
10/90

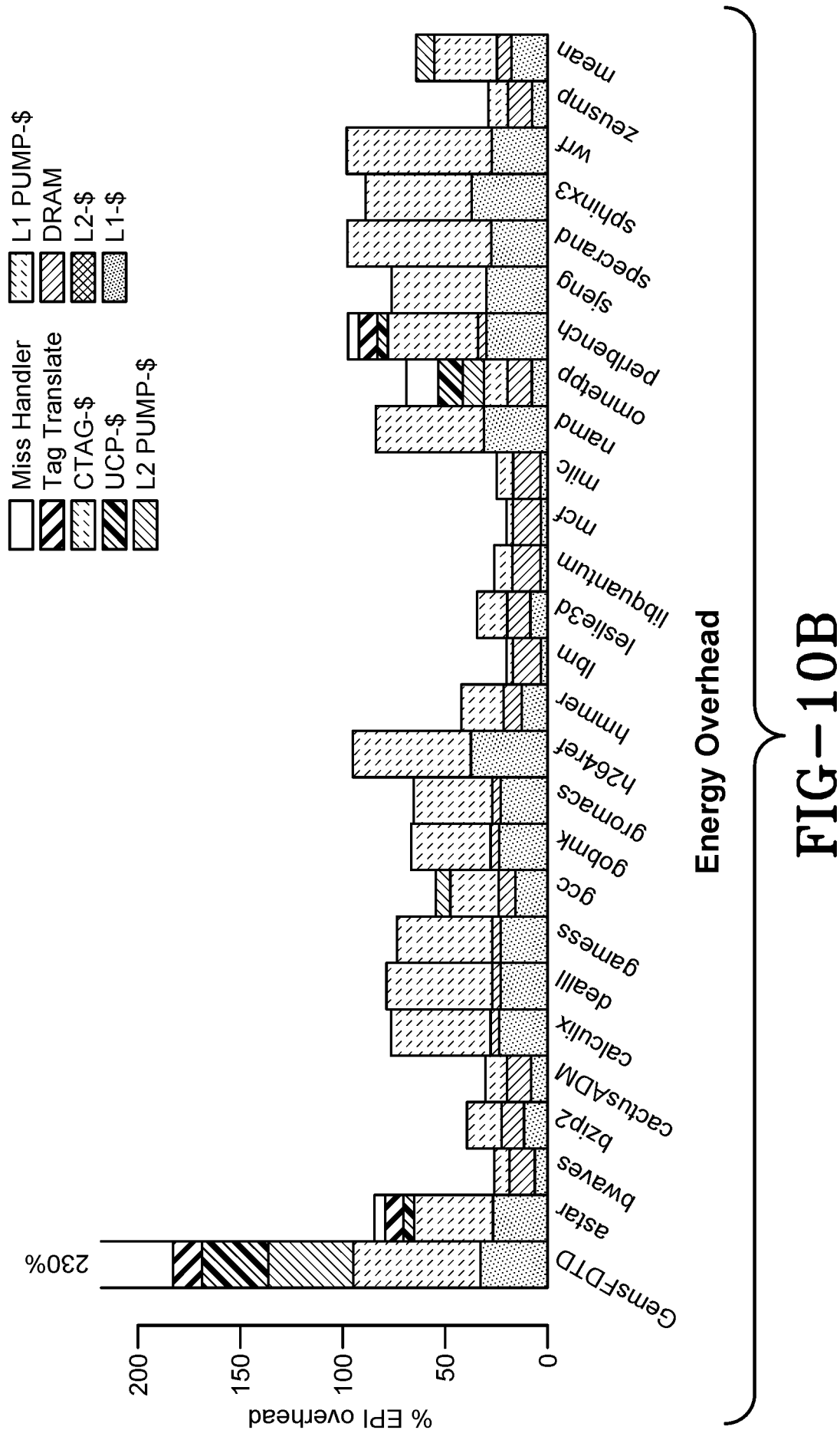
**FIG-9A****FIG-9B**

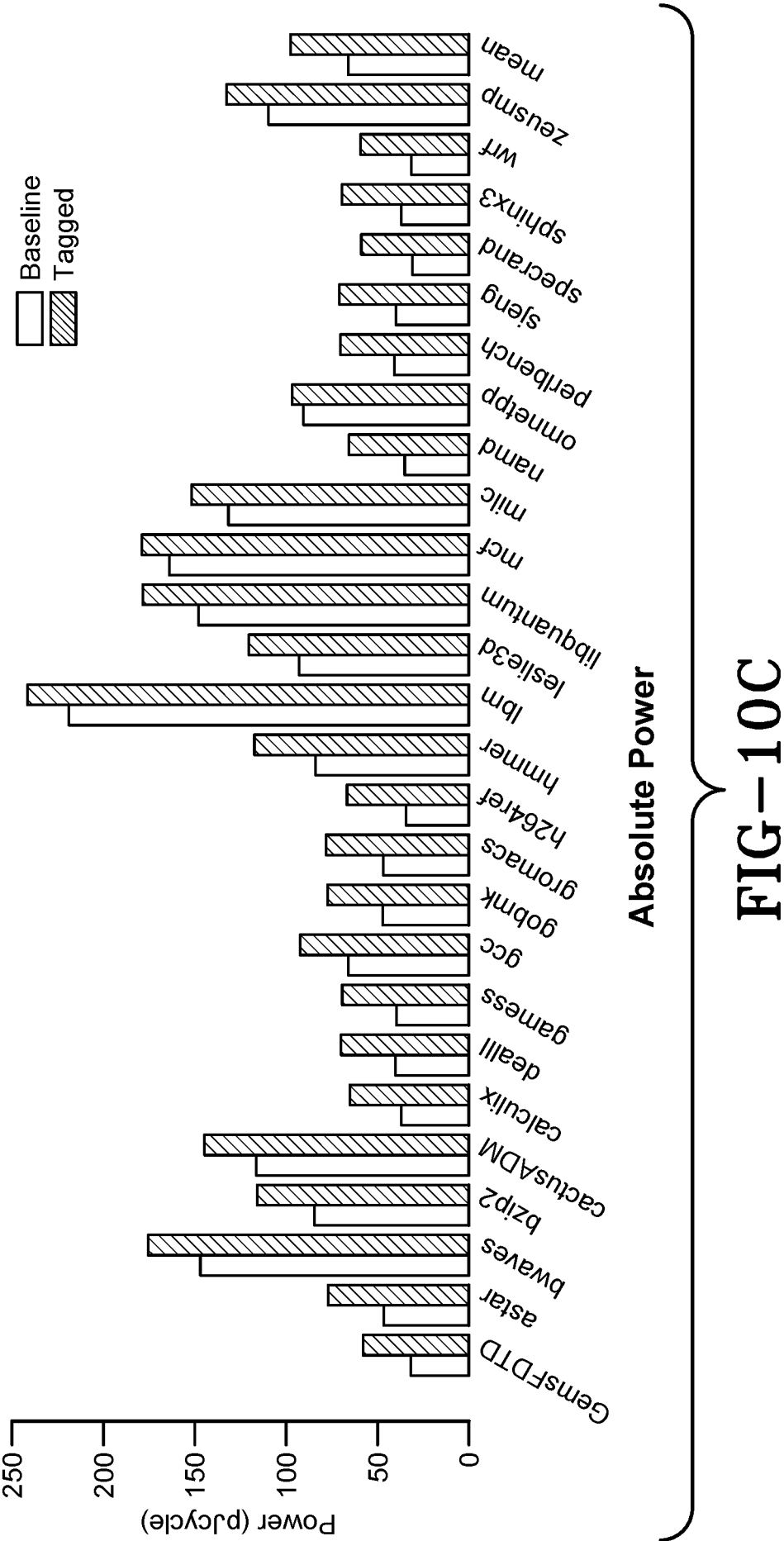
11/90

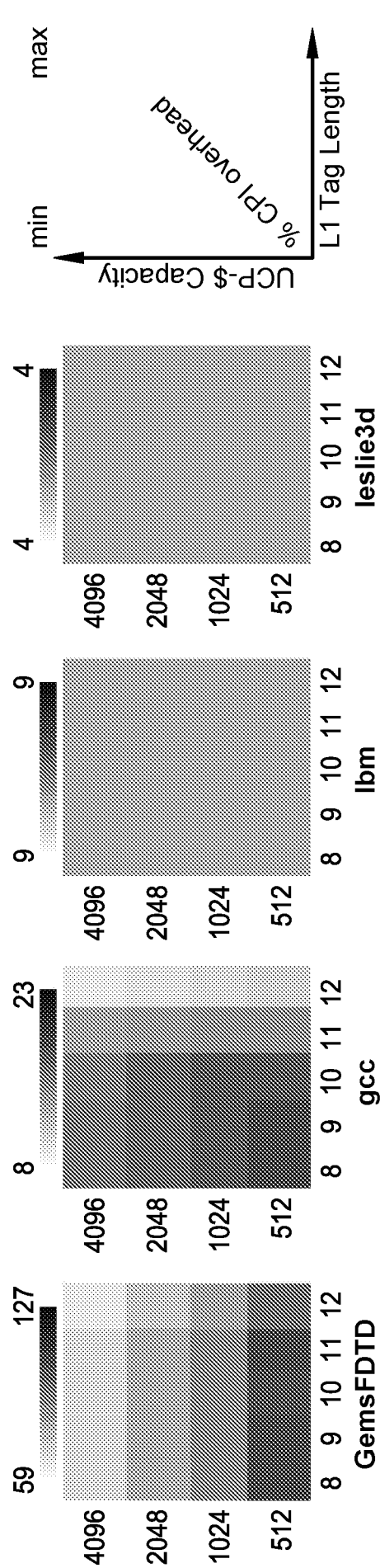
Miss Service Performance**FIG-9C****Miss-Handler CTAG-\$****FIG-9D**





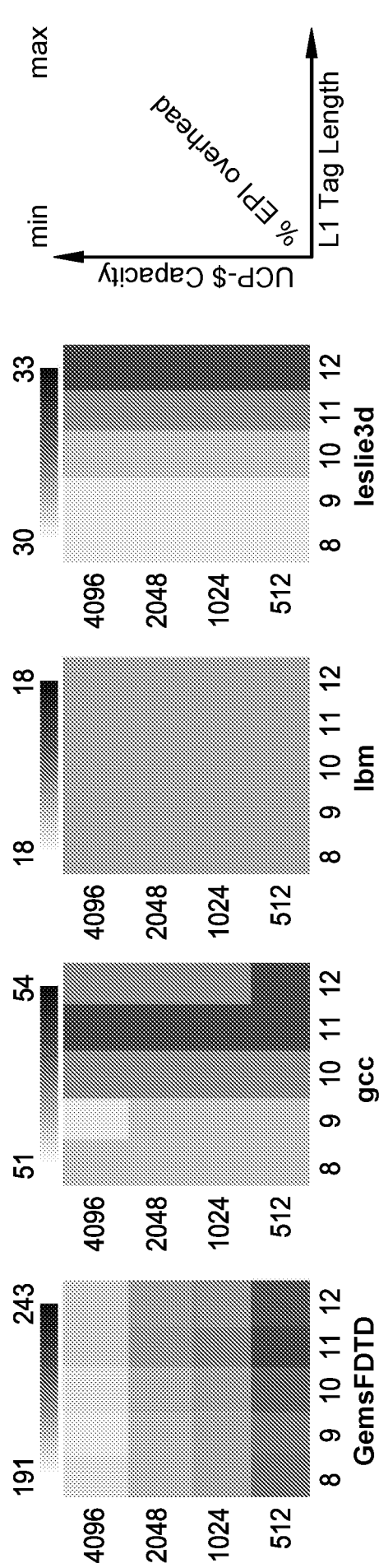






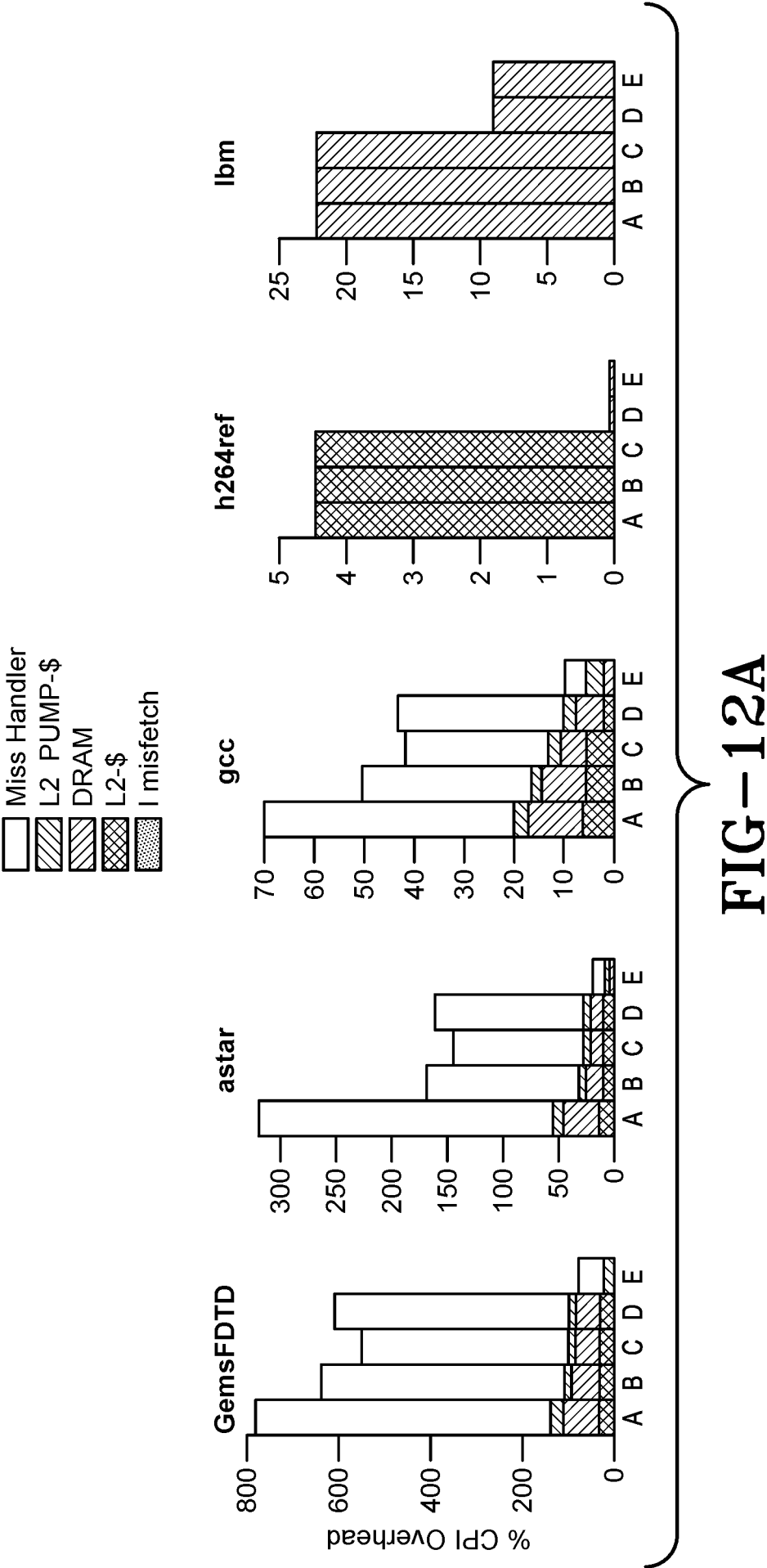
Runtime Overhead

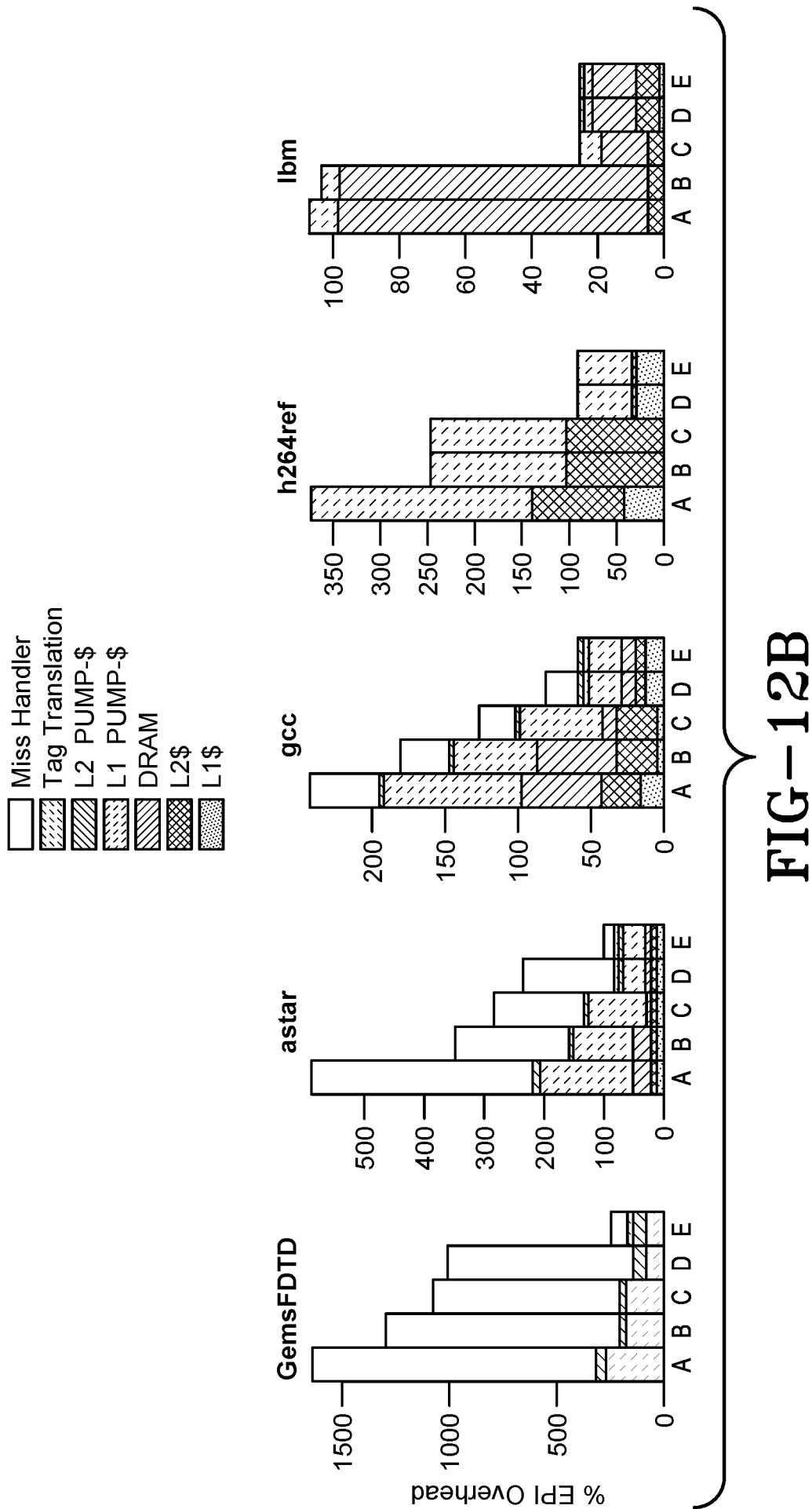
FIG-11A



Energy Overhead

FIG-11B





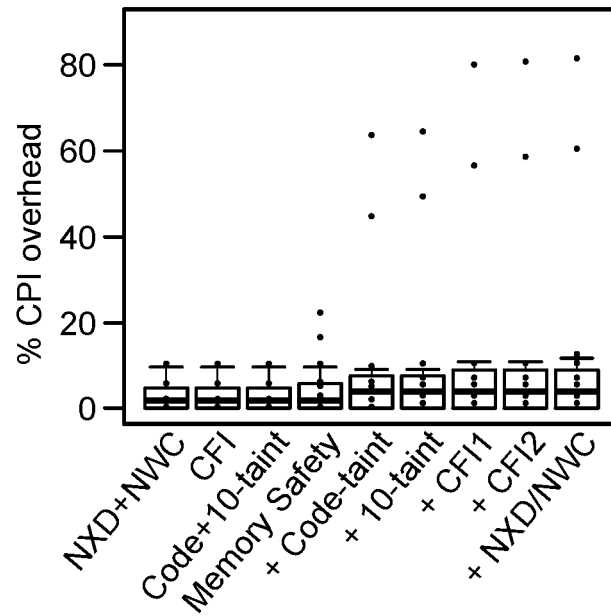
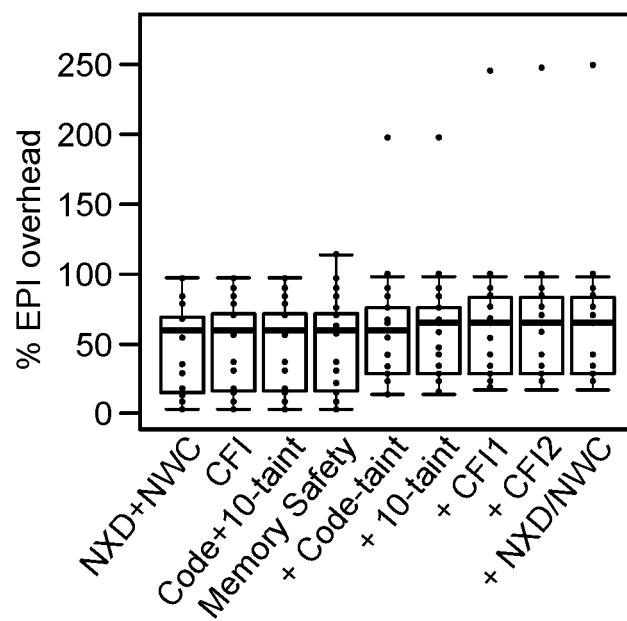
**FIG-13A****FIG-13B**

TABLE 1 - Summary of Investigated Policies

Policy	Threat	Metadata	Max Unique Tags	Tag Check (allow ?)	Tag Propagation Rules	Ref.
NXD+NWC	code injection	DATA or CODE on memory locations	2	$CI=CODE;$ $MR/=CODE$ on write	no	
Memory Safety	spatial/temporal memory safety violation on heap	color on pointers; region color + payload color on memory locations	$(\#mallocs)^2$	pointer color == referenced region color	$R \leftarrow OPj$ on mov/add/sub; $R \leftarrow payload(MR)$ on load	[17]
CFI	control-flow hijacking (JOP/ROP/code reuse)	unique id on each indirect control-flow source and target	$\#sources+\#targets$	$(PC,CI) \notin$ program's control-flow path	$PC \leftarrow CI$ on indirect jumps (including call, return)	[4]
Taint Tracking	untrusted code, low-integrity data from IO	source-taint-id on instructions and IO; set of input taint-ids on words	$2^{(\#code+IO\ ids)}$	user-defined check	$R \leftarrow CI \cup OP1 \cup OP2 \cup MR$	[47]
Composite	all of the above	structure with 4 elements	product of above	all of the above	all of the above (Alg.2 with $N=4$)	

FIG-14

TABLE 2 - Taxonomy of Tagging Schemes

Tag Bits	Propagate?	Outputs		Inputs		Usage
		allow?	R (result)	PC	PC CI OP1 OP2 MR	
2	X	soft	X	X	X	memory protection
word	X	limited prog.	X	X	X	memory hygiene, stack, isolation
32	X	limited prog.	X	X	X	unforgeable data, isolation
2	X	fixed	fixed	X	X	fine-grained synchronization
1	✓	fixed	X	X	X	capabilities
2-8	✓	fixed	fixed	X	✓	types
128	✓	fixed	copy	X	✓	memory safety
0	✓	software defined	X	propagate only one		invariant checking
1	✓	fixed	fixed	X	✓	taint
4	✓	limited programmability	X	X	✓	taint, interposition, fault isolation
10	✓	limited prog.	fixed	X	✓	taint, isolation
unspec.	✓	software defined	X	X	✓	flexible taint
32	✓	software defined	X	X	✓	programmable, taint, memory checking, reference counting
0-64	✓	software defined	✓	✓	✓	information flow, types
Unbounded	✓	software defined	✓	✓	✓	fully programmable, pointer-sized tags

Propagate = tag propagates with data (✓) or is a property of memory address (X)

allow? = logic for allowing/disallowing operations

R = policy impacts tag of result

PC = programmable tag on program counter

CI = is current instruction tag checked or propagated

OP1, OP2, MR = is tag on these inputs checked or propagated

FIG-15

TABLE 3 - Memory Resource Estimates for the Baseline and Simple PUMP-extended Processors at 32nm node

Unit	Design	Organization	Area (mm ²)	Access Energy Read/Write (pJ)	Static Power (pJ/cyc)	Latency (ps)	Cyc
Register File (Int. and FP)	Baseline	64b, 2R1W, {32 Integer, 32 Floating}	0.002	0.3/0.5	0.08	264	1
	64b tag extended	128b, 2R1W, {48 Integer, 32 Floating}	0.007	1.0/1.4	0.23	295	1
L1-\$ (I and D)	Baseline	64KB, 4-way, 64B/line	0.236	17/11	14.4	880	1
	64b tag extended	64KB, 4-way, 128B/line (eff. 32KB, 64B/line)	0.244	19/14	14.5	880	1
L2-\$ (unified)	Baseline	512KB, 8-way, 64B/line	1.207	393/481	0.111	4000	5
	64b tag extended	1MB, 8-way, 128B/line (eff. 512KB, 64B/line)	2.350	758/1223	0.214	4930	5
TLB (I and D)	Either	1KB, 2-way set assoc.	0.04	3.6/4.5	2	800	1
DRAM	Baseline	1GB, access 64B line per transfer	-	15,000	-	-	100
	64b tag extended	1GB, access 128B line (eff. 64B line per transfer)	-	31,000	-	-	130
L1 PUMP-\$	64b tag	fully associative 1024 entry, 328b match, 128b out (not used; shown only for reference)	1.500	750/900	-	3000	4
L2 PUMP-\$	64b tag	multi-hash 1024 entry, 328b match, 128b out	0.683	51/62	32	500	1
		multi-hash 4096 entry, 328b match, 128b out	0.994	173/444	0.085	3300	4

Total Baseline Area 1.485mm²**Total 64b-tagged Area 4.318mm² (+ 190% over baseline)****FIG-16**

24/90

TABLE 4 - PUMP Parameter Range Table

Unit	Parameter	Range	Final
L1 PUMP-\$	Capacity	512-4096	1024
	Tag Bits	8-12	10
L2 PUMP-\$	Capacity	1024-8192	4096
	Tag Bits	13-16	14
UCP-\$	Capacity	512-4096	2048
CTAG-\$	Capacity	128-1024	512

FIG-17

TABLE 5 - Memory Resource Estimates for the PUMP-Optimized Processors at 32nm node

Unit	Design	Organization	Area (mm ²)	Access Energy Read/Write (pJ)	Static Power (pJ/cyc)	Latency (ps)	Cyc
Register File	Extended 10b	74b, 2R1W, {48 Integer, 32 Floating}	0.005	0.4/0.5	0.13	360	1
L1 Cache	10b-tag	74KB, 4-way, 74B/line (eff. 64KB, 64B/line)	0.272	19/13	16.4	975	1
L2 Cache	14b-tag	592KB, 8-way, 78B/line (eff. 512KB, 64B/line)	1.247	343/640	0.133	4600	5
TLB	-	1KB, 2-way set-associ.	0.040	3.6/4.5	2	800	1
DRAM	64b-tag	1GB, access 128B line (move 76B)	-	17,500	-	-	112
L1 PUMP Cache	10b L1 tag	multihash 1024-entry, 58b match, 20b out	0.095	15/43.2	2.22	520	1
L2 PUMP Cache	14b L2 tag	multihash 4096-entry, 78b match, 28b out	0.287	99.4/267	0.032	2800	3
full → L2-tag	64b → 14b	8K entry, 64b match, 14b out, multihash	0.432	166/436	0.052	3400	4
L2-tag → full	14b → 64b	16K x 64 SRAM	0.216	55.5/31.5	0.027	1700	2
L2-tag → L1-tag	14b → 10b	16K x 11 SRAM	0.038	8.8/4.7	0.004	1420	2
L1-tag → L2-tag	10b → 14b	1K x 14 SRAM	0.004	0.8/1	0.0004	780	1
UCP Cache	64b tags	multihash 2048-entry, 328b match, 128b out	0.377	196/479	0.035	2730	3
CTAG Cache	64b tags	multihash 512-entry	0.107	56/139	0.009	1700	2

Total Area 3.120mm² (+ 110% over baseline)

FIG-18

Algorithm 1 Taint Tracking Miss Handler (Fragment)

```

1: switch (op)
2: case add, sub, or:
3:    $PC_{new} \leftarrow PC$ 
4:    $R \leftarrow \text{canonicalize}(CI \cup OP1 \cup OP2)$ 
5:   (... cases for other instructions omitted ...)
6: default: trap to error handler
  
```



FIG-19

Algorithm 2 N - Policy Miss Handler

```

1: for  $i=1$  to  $N$  do
2:    $M_i \leftarrow \{op, PC[i], CI[i], OP1[i], OP2[i], MR[i]\}$ 
3:    $\{pc_i, res_i\} \leftarrow \text{policy}_i(M_i)$ 
4:    $PC_{new} \leftarrow \text{canonicalize}([pc_1, pc_2, \dots, pc_N])$ 
5:    $R \leftarrow \text{canonicalize}([res_1, res_2, \dots, res_N])$ 
  
```



FIG-20

27/90

Algorithm 3 N - Policy Miss Handler with HW support

```

1: for i=1 to N do
2:    $M_i \leftarrow \{op, PC[i], CI[i], OP1[i], OP2[i], MR[i]\}$ 
3:    $\{hit, pc_i, res_i\} \leftarrow UCP\$ (i, M_i)$ 
4:   if !hit then
5:      $\{pc_i, res_i\} \leftarrow policy_i (M_i)$ 
6:     insert  $\{i, pc_i, res_i\}$  into UCP-$
7:    $pc[1...N] \leftarrow \{pc_1, pc_2, ..., pc_N\}$ 
8:    $\{hit, PC_{new}\} \leftarrow CTAG\$ (pc[1...N])$ 
9:   if !hit then
10:     $PC_{new} \leftarrow canonicalize (pc[1...N])$ 
11:    insert  $\{pc[1...N], PC_{new}\}$  into CTAG-$
12:    $res[1...N] \leftarrow \{res_1, res_2, ..., res_N\}$ 
13:    $\{hit, R\} \leftarrow CTAG\$ (res[1...N])$ 
14:   if !hit then
15:     $R \leftarrow canonicalize (res[1...N])$ 
16:    insert  $\{res[1...N], R\}$  into CTAG-$
  
```


FIG-21

PUMP Rule Cache Dataflow and Microarchitecture

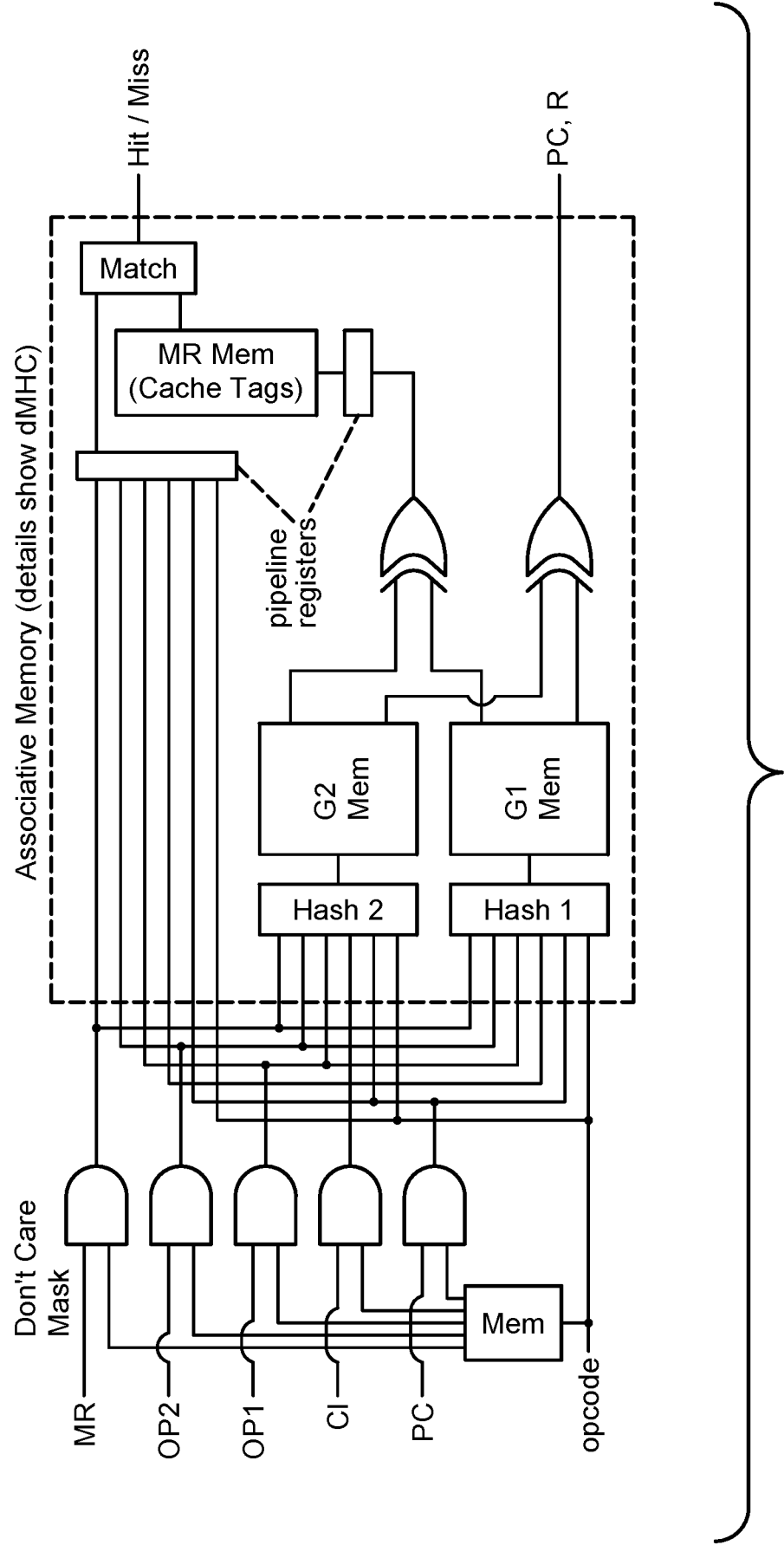


FIG-22

PUMP Microarchitecture

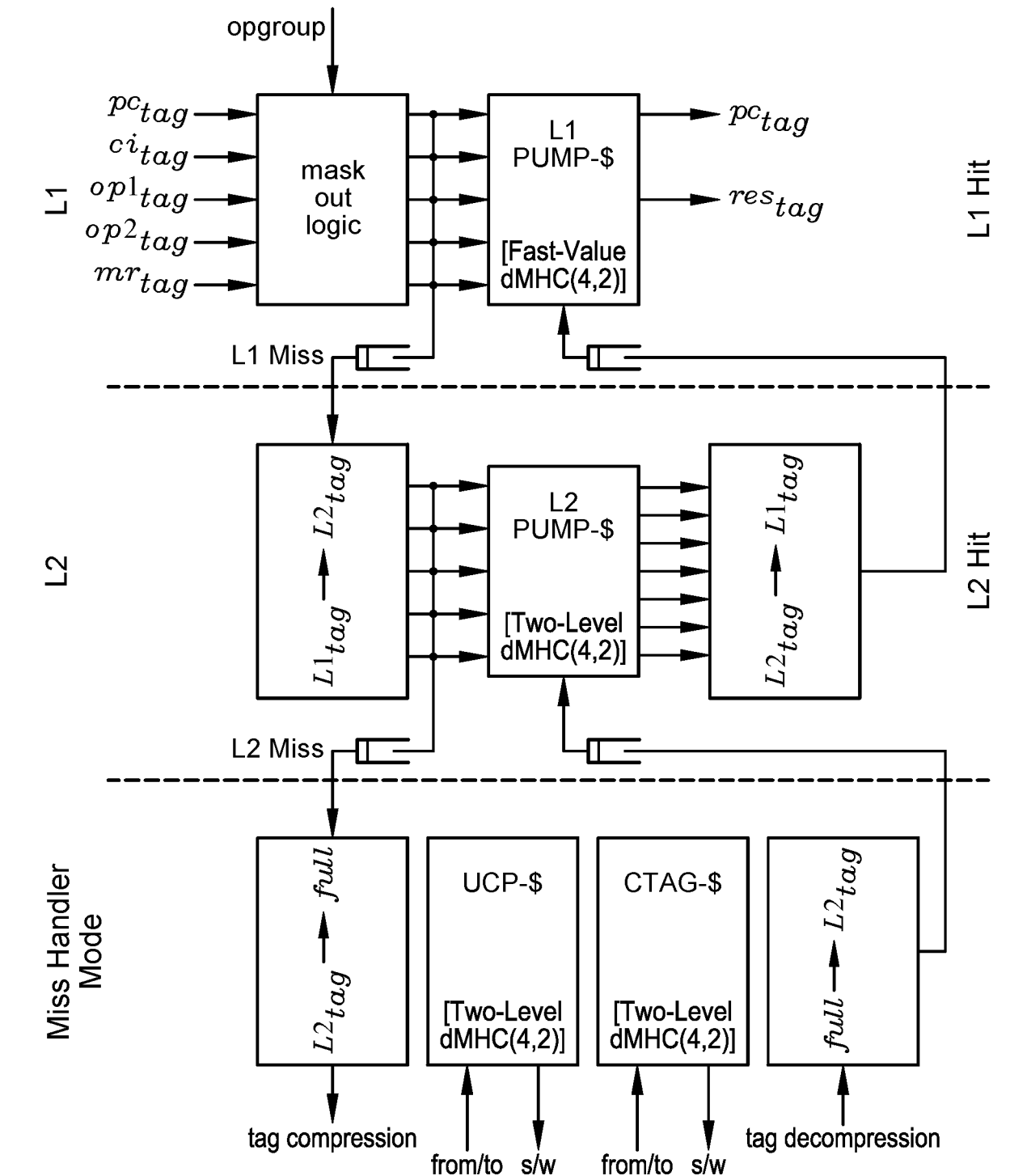
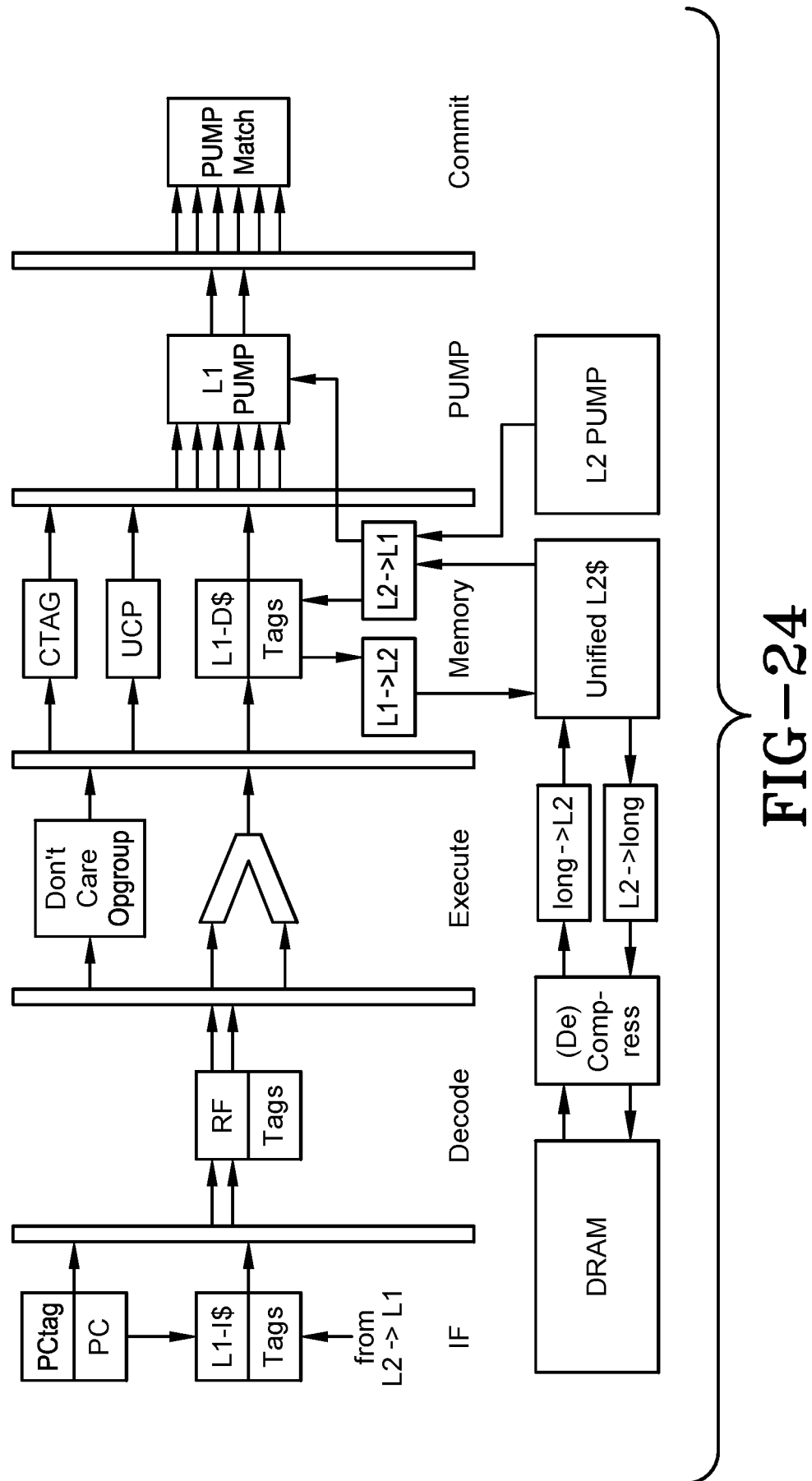


FIG-23



31/90

900	902	904	906	908
Address	Privilege	Name	Description	
901a 0x5C0	SRW	sboottag	Bootstrap Tag for OS (cleared on read)	
901b 0x5C1	SRW	spubuntrust	Public, Untrusted Tag Encoding	
901c 0x5C2	SRW	sdefaulttag	Default Tag	
901d 0x5C8	SRW	sopgrpaddr	Address for write to opgroup/care table ((opcode<<3) funct3)	
901e 0x5C9	SRW	sopgrpvalue	Data for write to opgroup/care table;	
901f 0x5CC	SRW	spumpflush	write to this CSR triggers insertion into table Write to this CSR triggers PUMP flush	
901g 0x5D0	SRW	sopgrp	Opgroup for rule miss	
901h 0x5D1	SRW	spctag	PC tag for rule miss	
901i 0x5D2	SRW	scitag	CI (current instruction) tag for rule miss	
901j 0x5D3	SRW	srs1tag	RS1 tag for rule miss	
901k 0x5D4	SRW	srs2tag	RS2 tag for rule miss	
901l 0x5D5	SRW	srs3tag	RS3 tag for rule miss	
901m 0x5D6	SRW	smtag	memory tag for rule miss	
901n 0x5D7	SRW	sfunct12	funct12 from instruction	
901o 0x5D8	SRW	ssubinstr	when a single tag applies to multiple instructions, which one are we executing? (a few bits of the PC) (when instruction length=32, tagged word length=64, this is odd/even)	
901p 0x5E0	SRW	snewpctag	Tag to place on PC on completion of instruction	
901q 0x5E1	SRW	srtag	Tag to place on RD register or memory result from instruction; write to this CSR triggers write of rule to PUMP	
901r 0x7C0	MRW	mtagmode	Mode for PUMP operation	

FIG. 25

910 ↗

	912 mtagmode Encoding	914 Operation	916 Tag Results
911a	000	off	not updated
911b	010	write default tag on all results	defaulttag
911c	100	PUMP engaged and operational U only	default prop. or PUMP output
911d	101	PUMP engaged and operational S, U	default prop. or PUMP output
911e	110	PUMP engaged and operational H, S, U	default prop. or PUMP output
911f	111	PUMP engaged and operational M, H, S, U	default prop. or PUMP output

FIG. 26

33/90

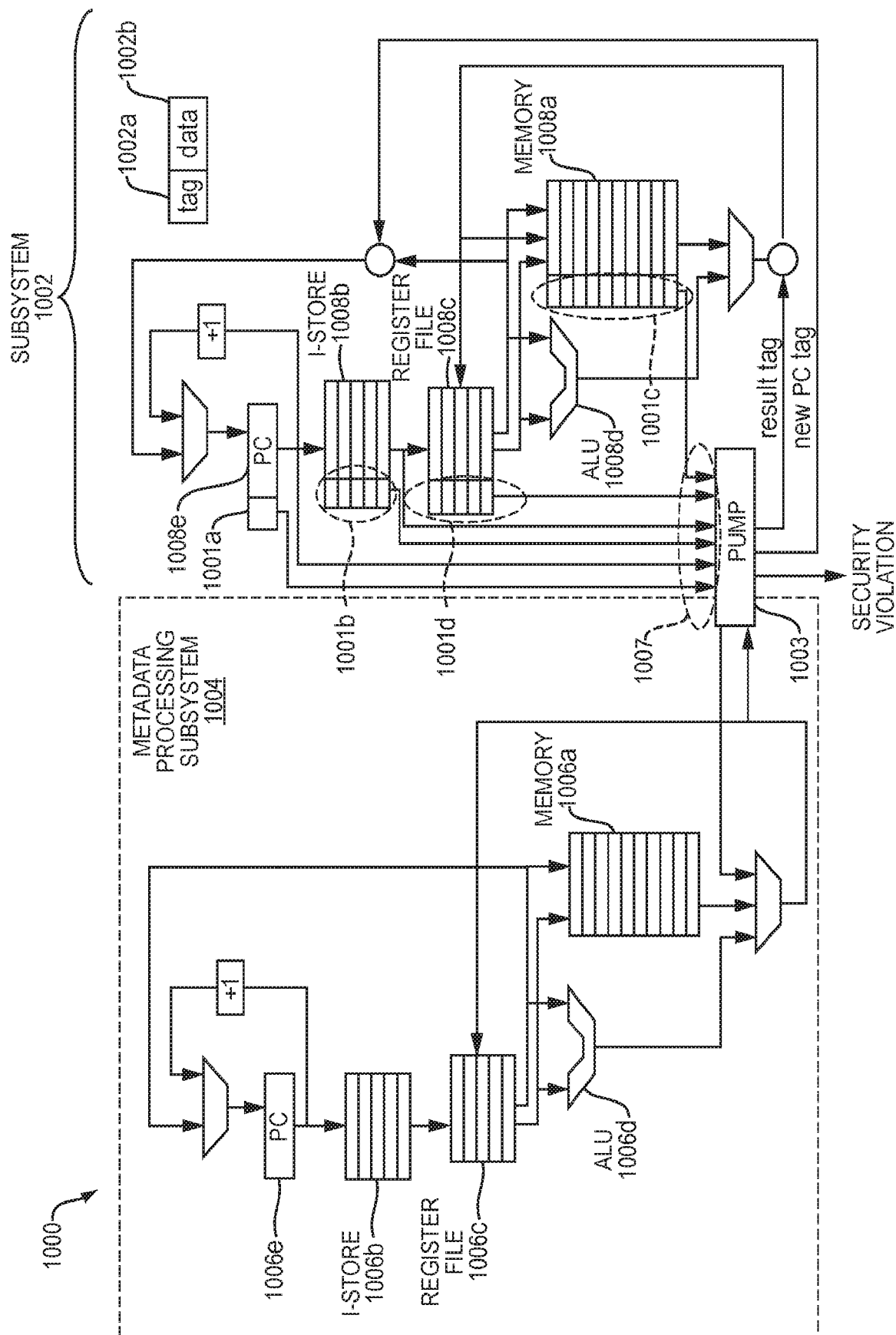


FIG. 27

34/90

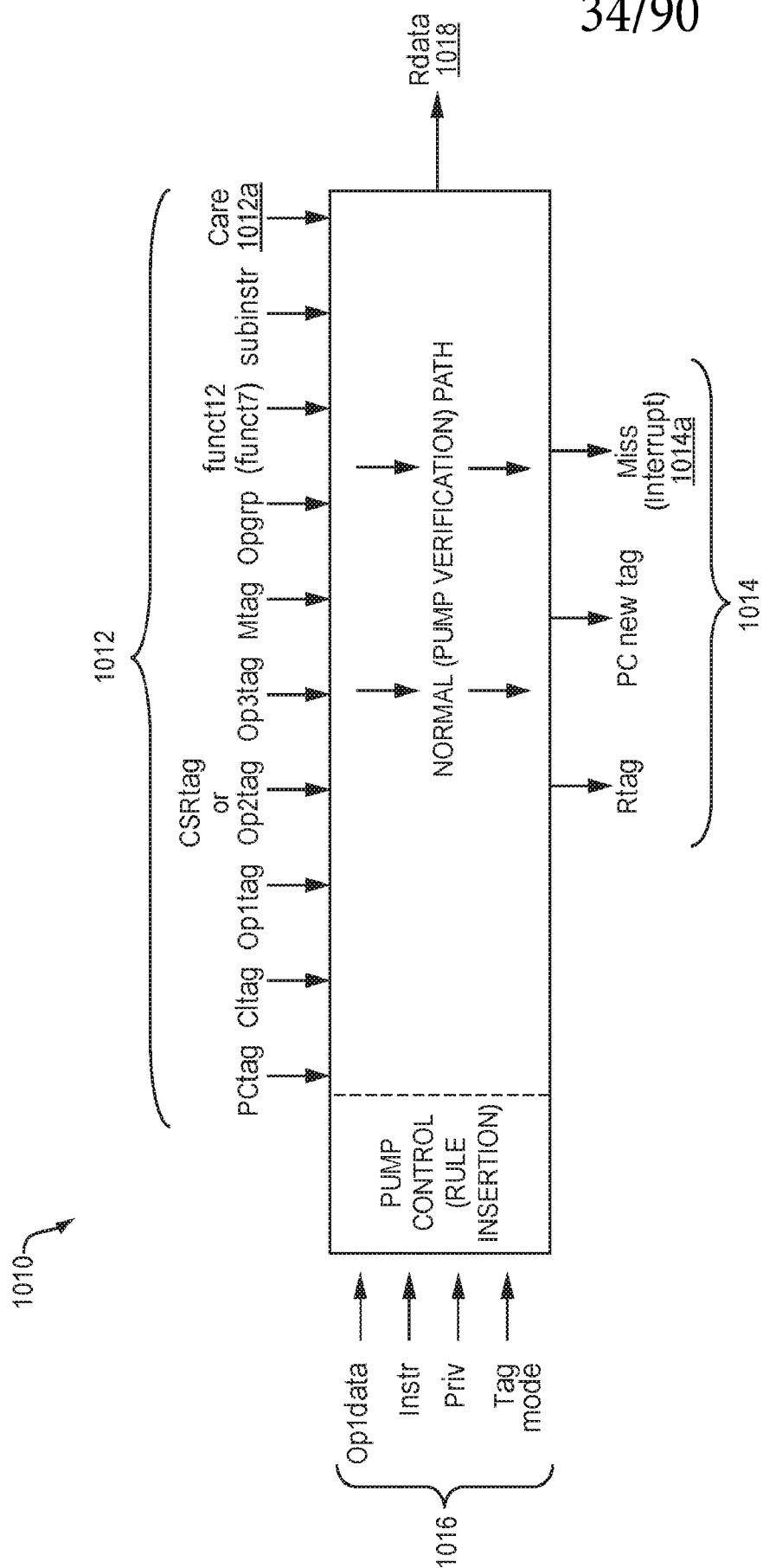


FIG. 28

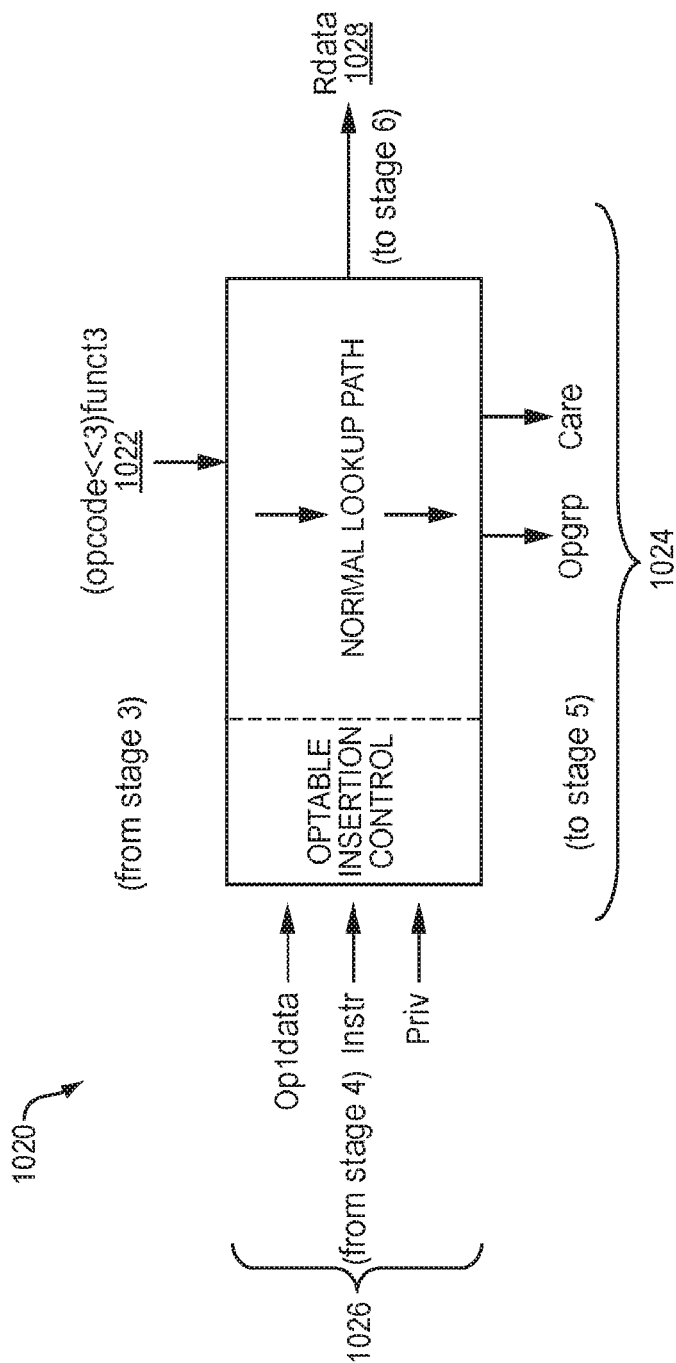


FIG. 29

1030

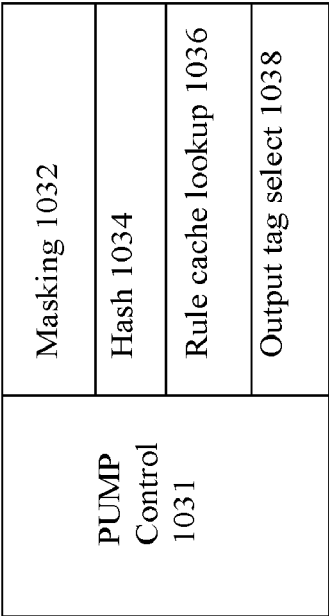
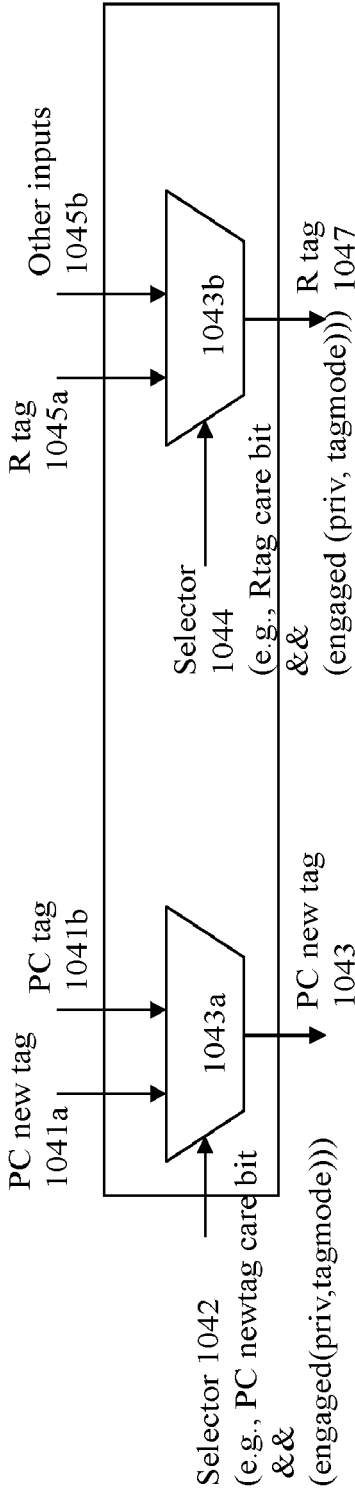


FIG. 30

1040



1049

Engaged =(((priv==U) && (tagmode in (PUMP_U, PUMP_SU, PUMP_HSU, PUMP_MHSU)))
||((priv==S) && (tagmode in (PUMP_SU, PUMP_HSU, PUMP_MHSU)))
||((priv==H) && (tagmode in (PUMP_HSU, PUMP_MHSU)))
|| (prive==M) && (tagmode == PUMP_HSU)))

FIG. 31

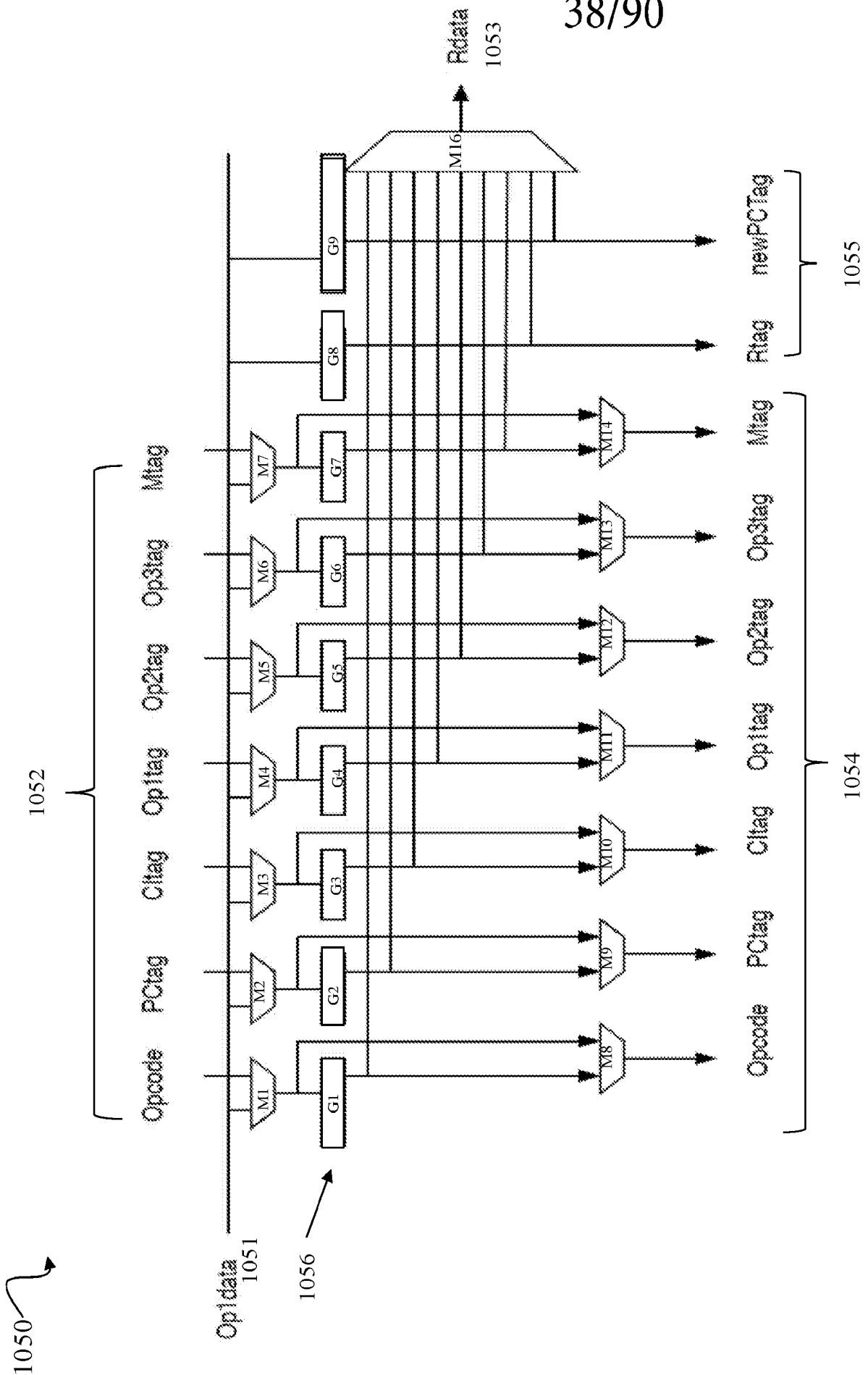
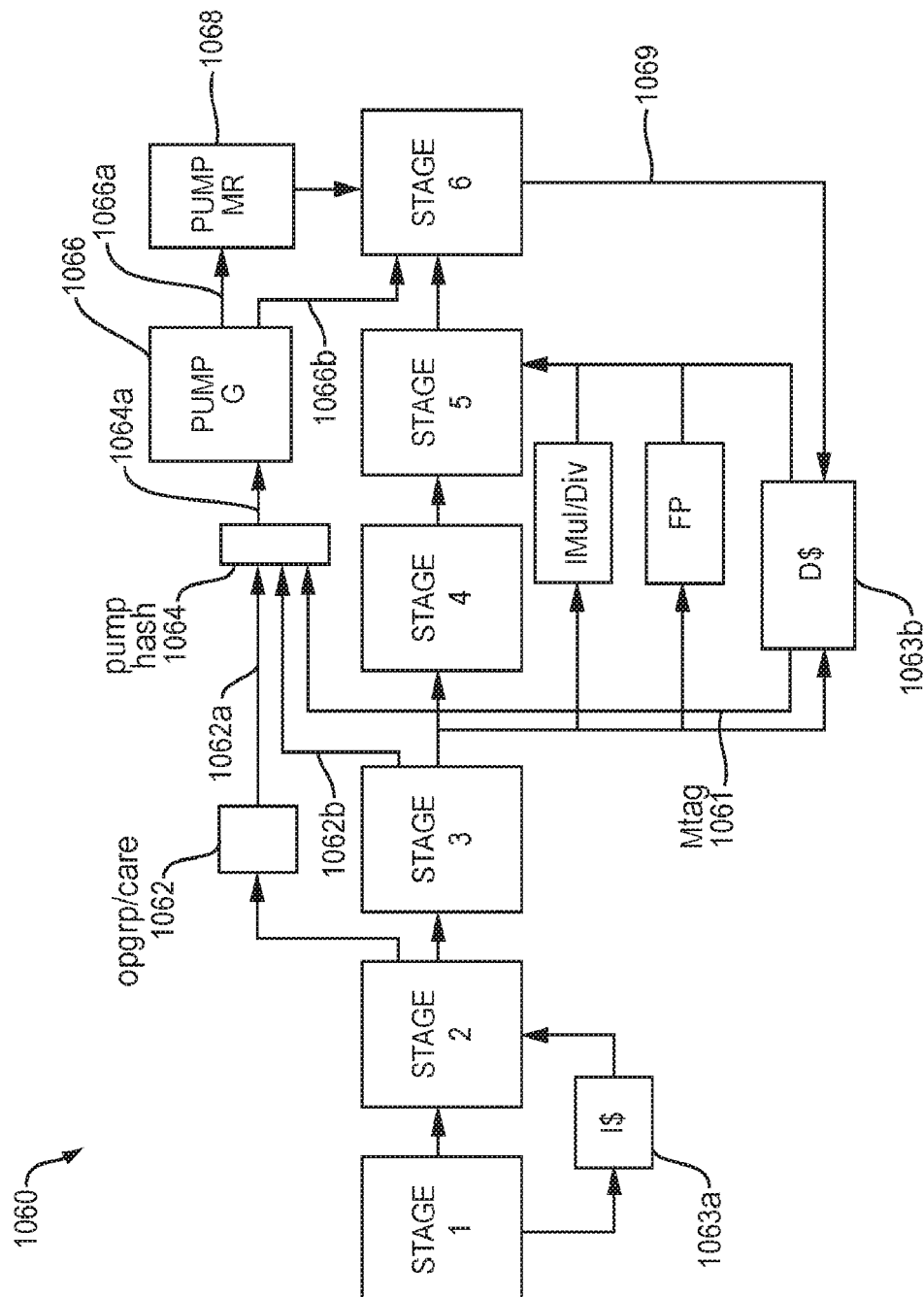


FIG. 32

39/90



354

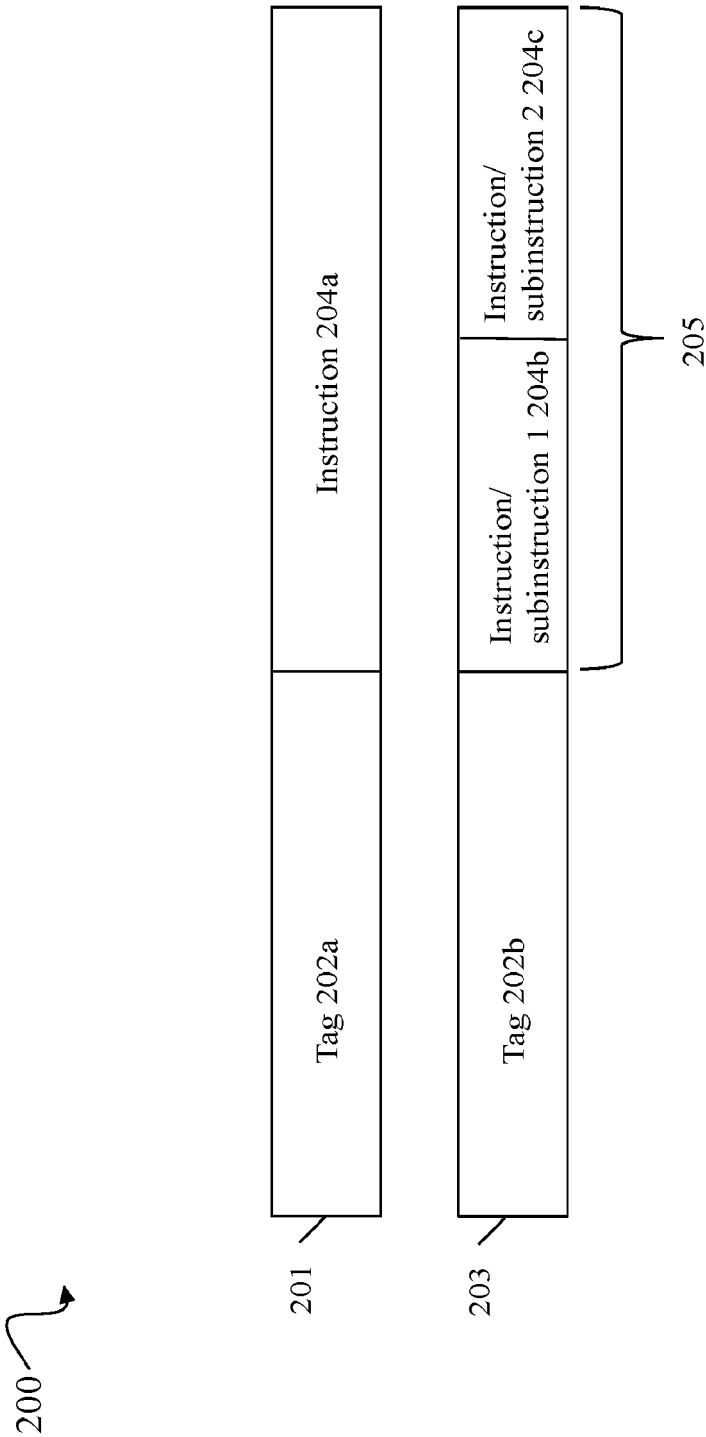


FIG. 34

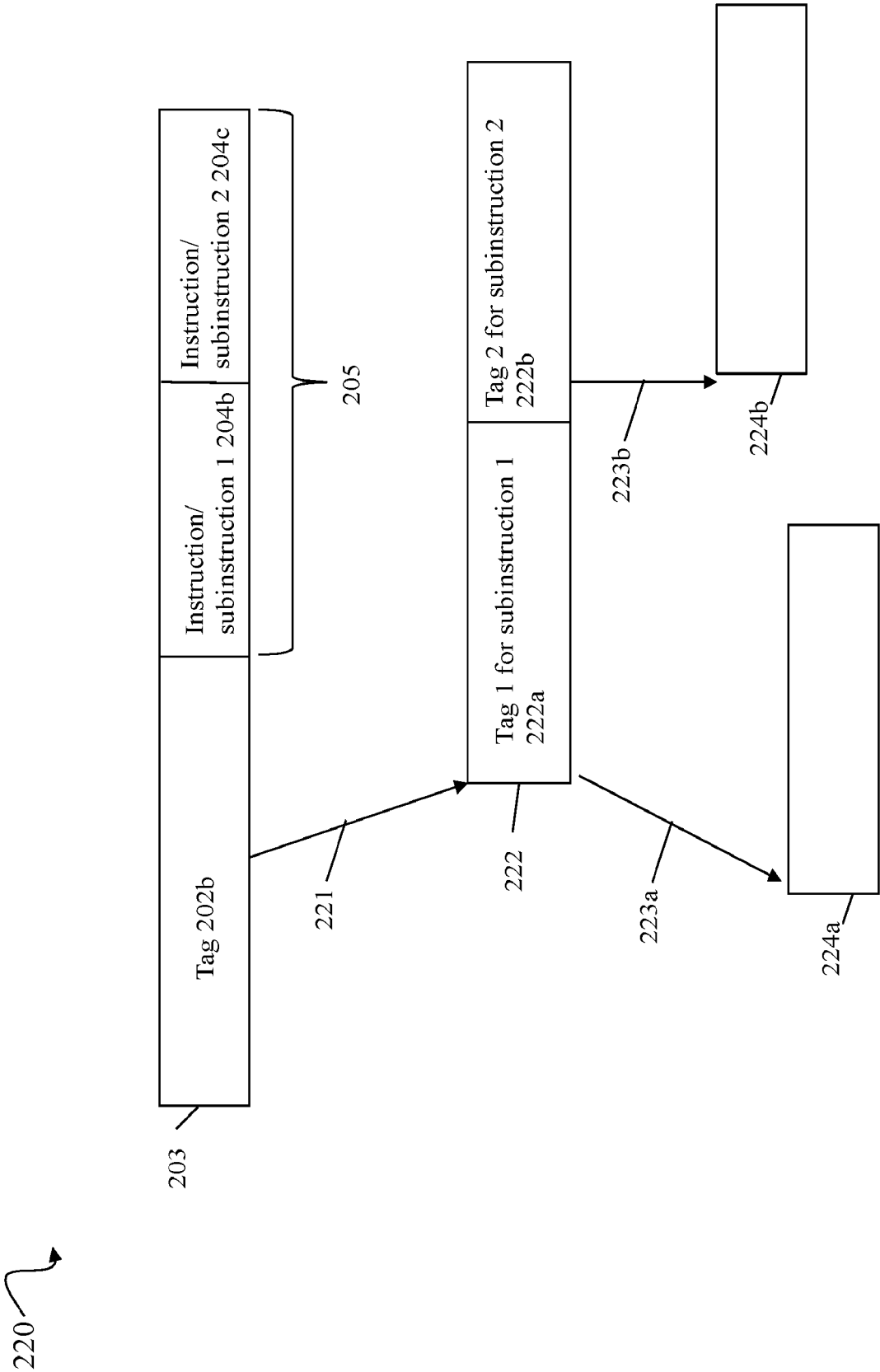


FIG. 35

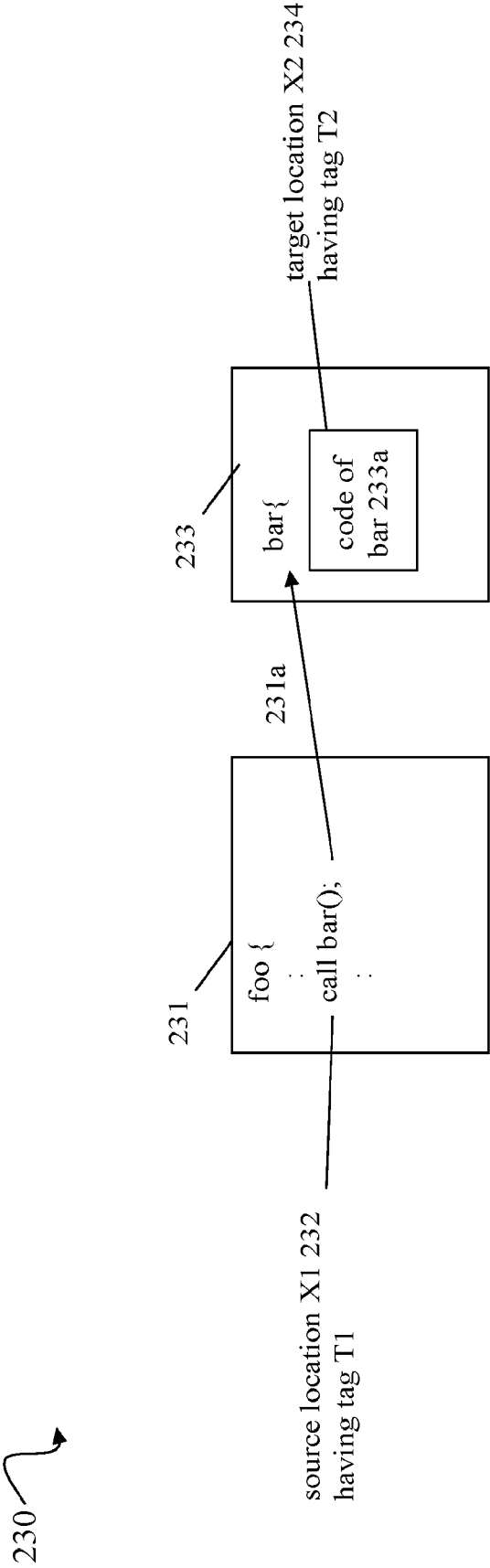


FIG. 36

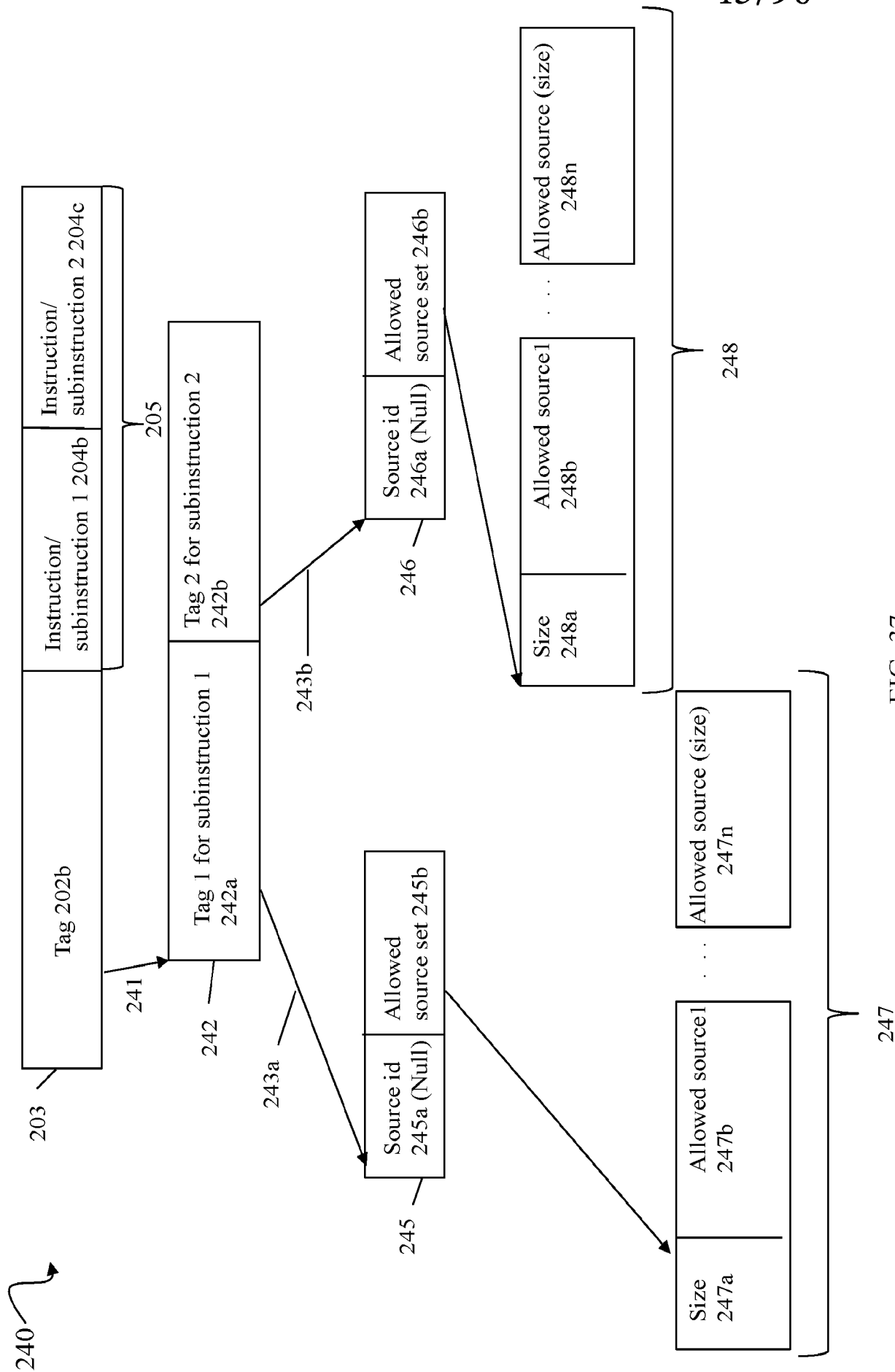


FIG. 37

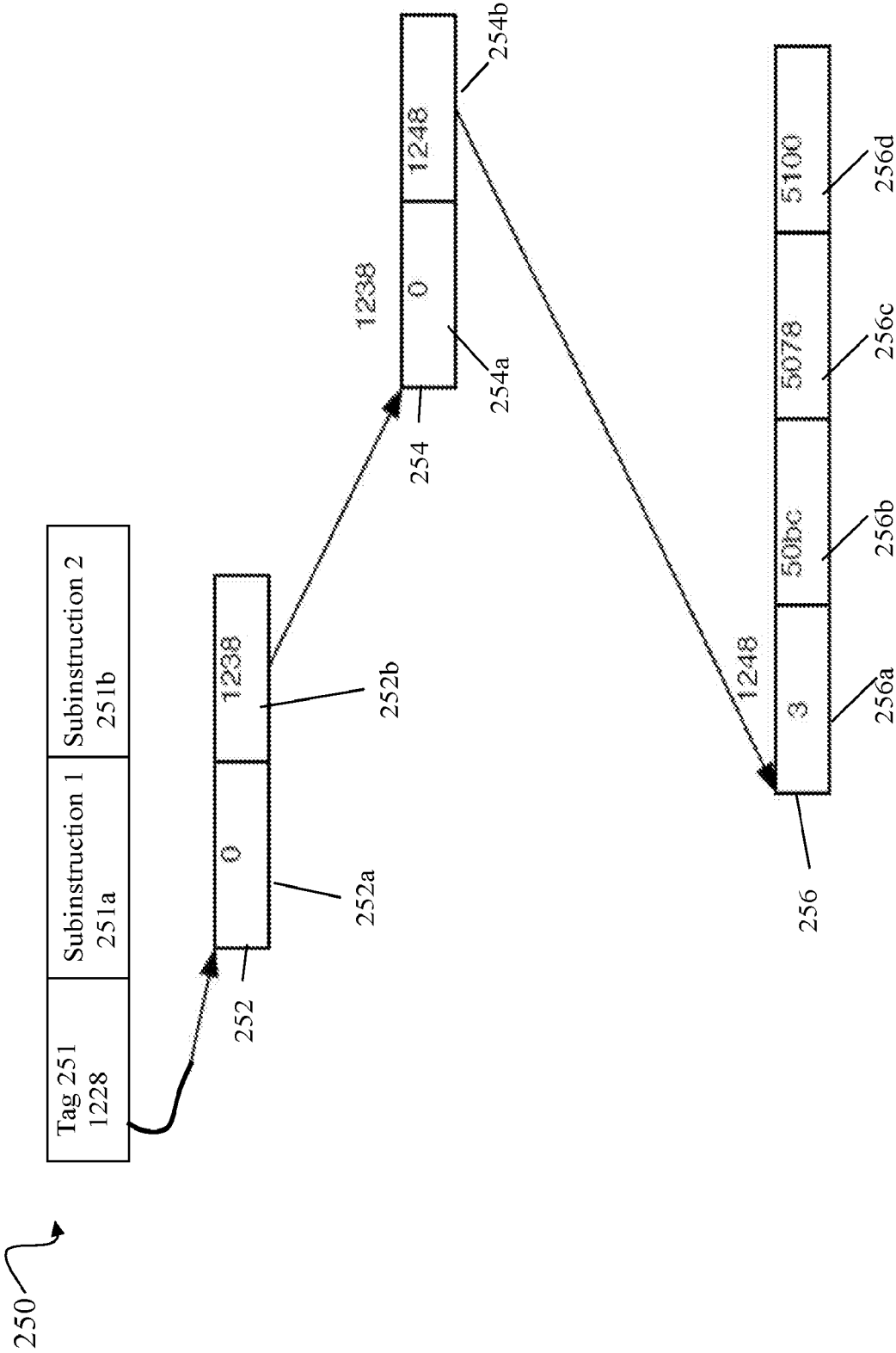


FIG. 38

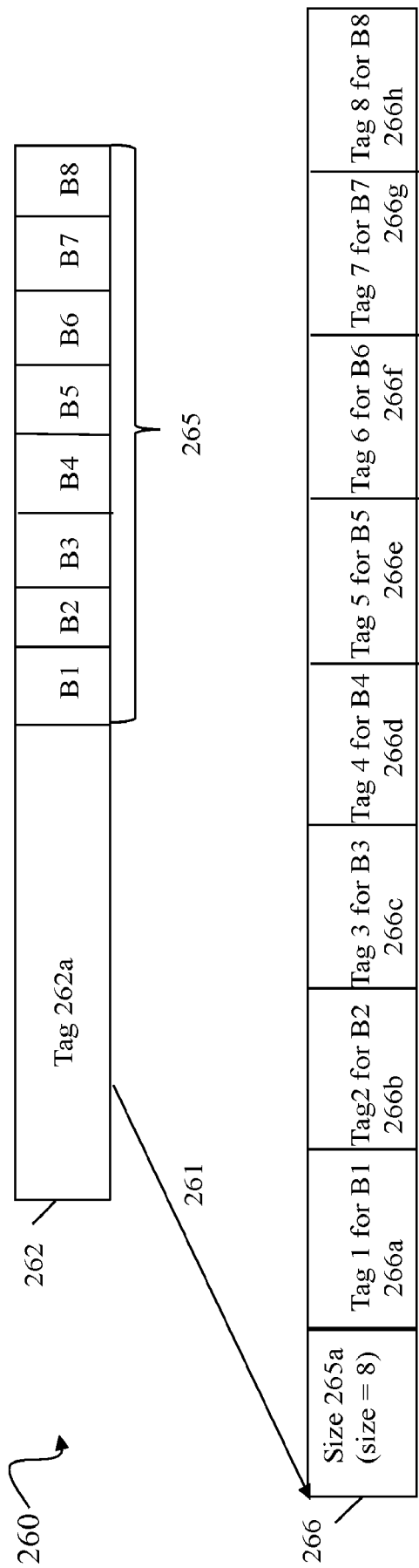


FIG. 39

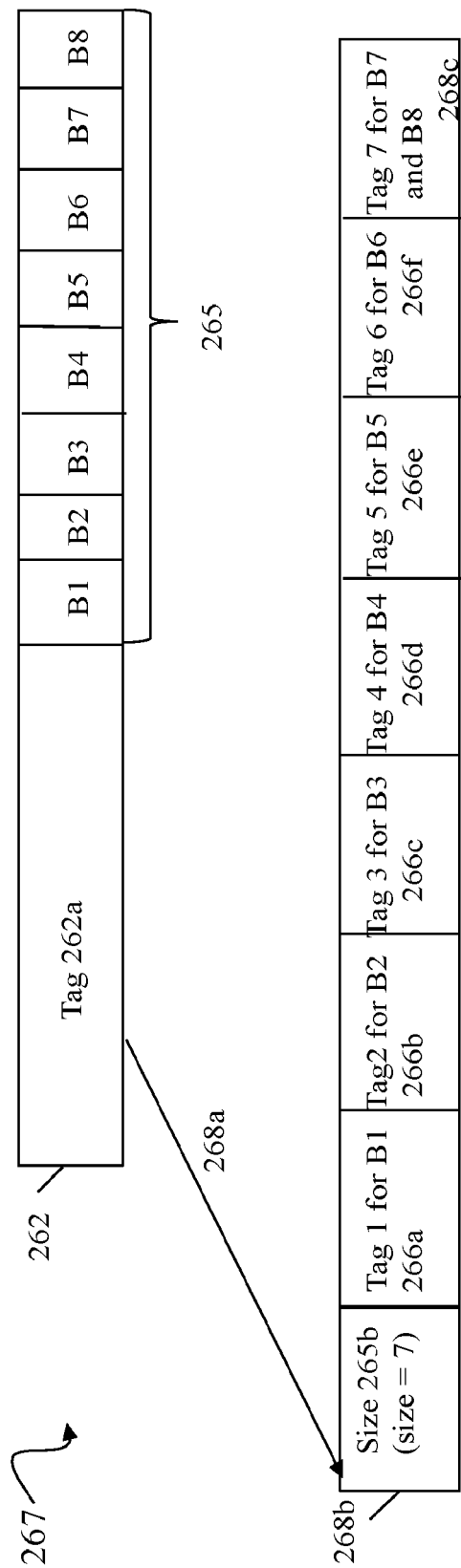


FIG. 40

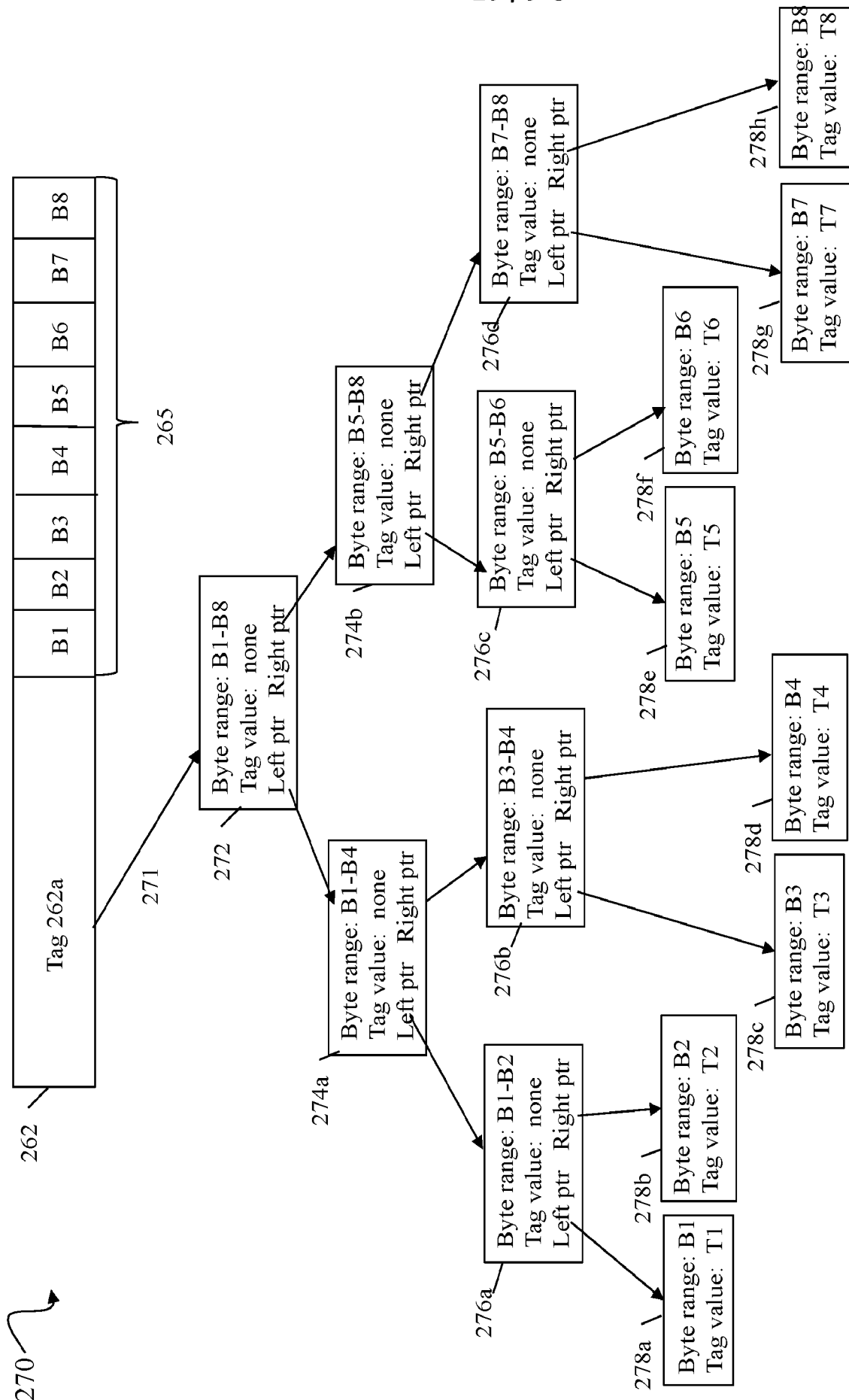


FIG. 41

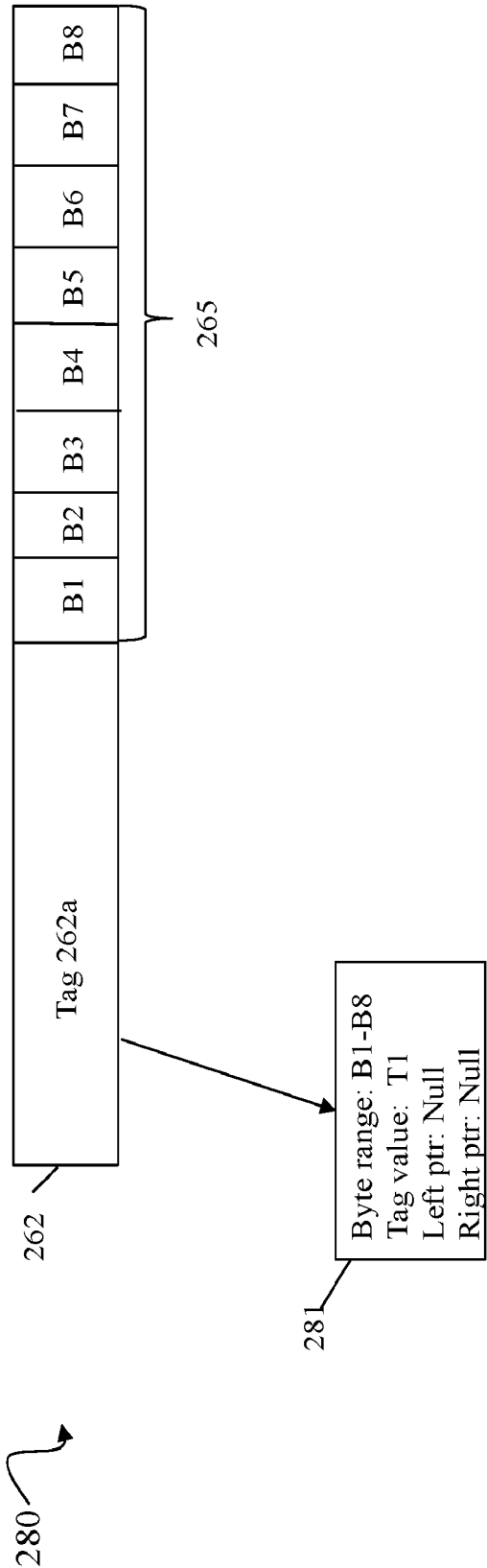


FIG. 42

400 ↗

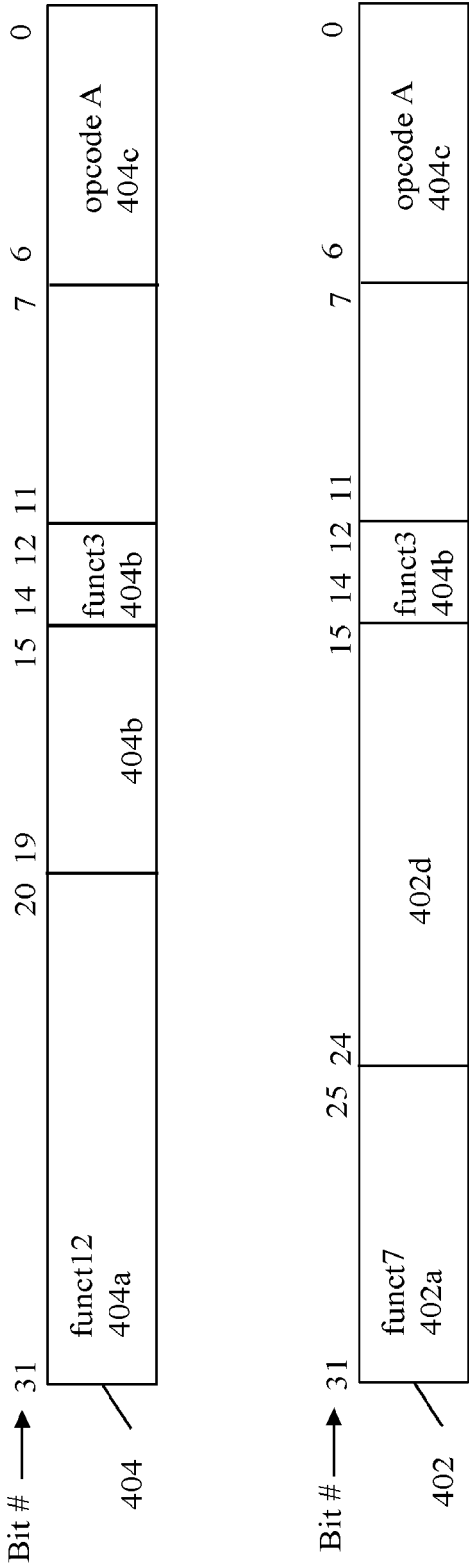


FIG. 43

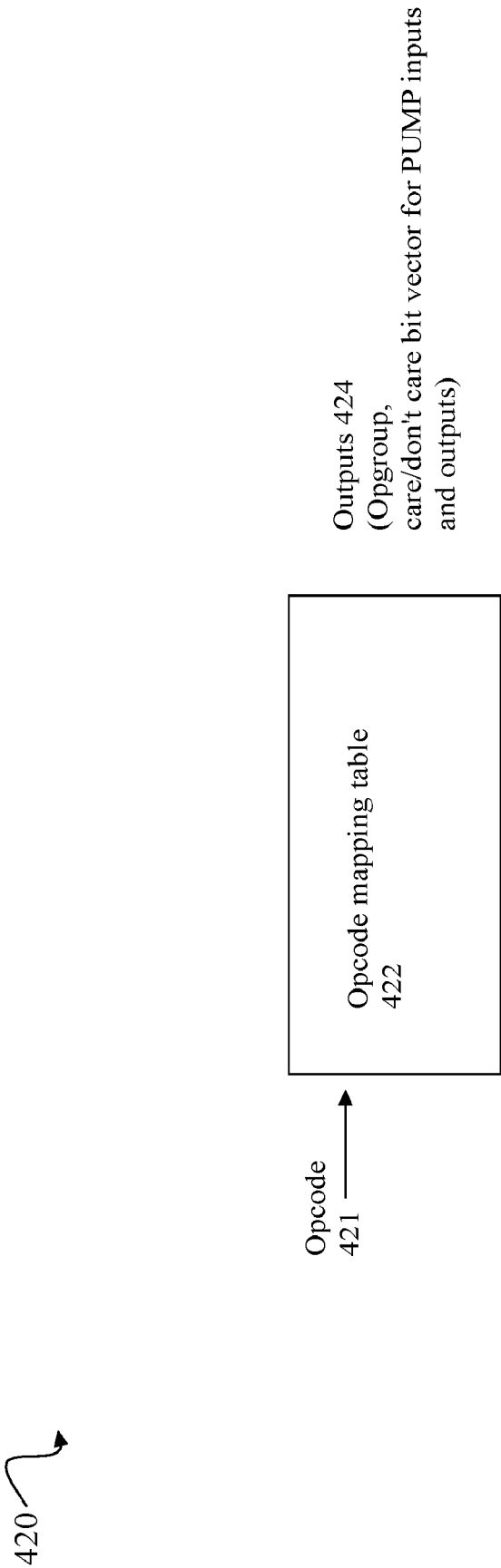


FIG. 44

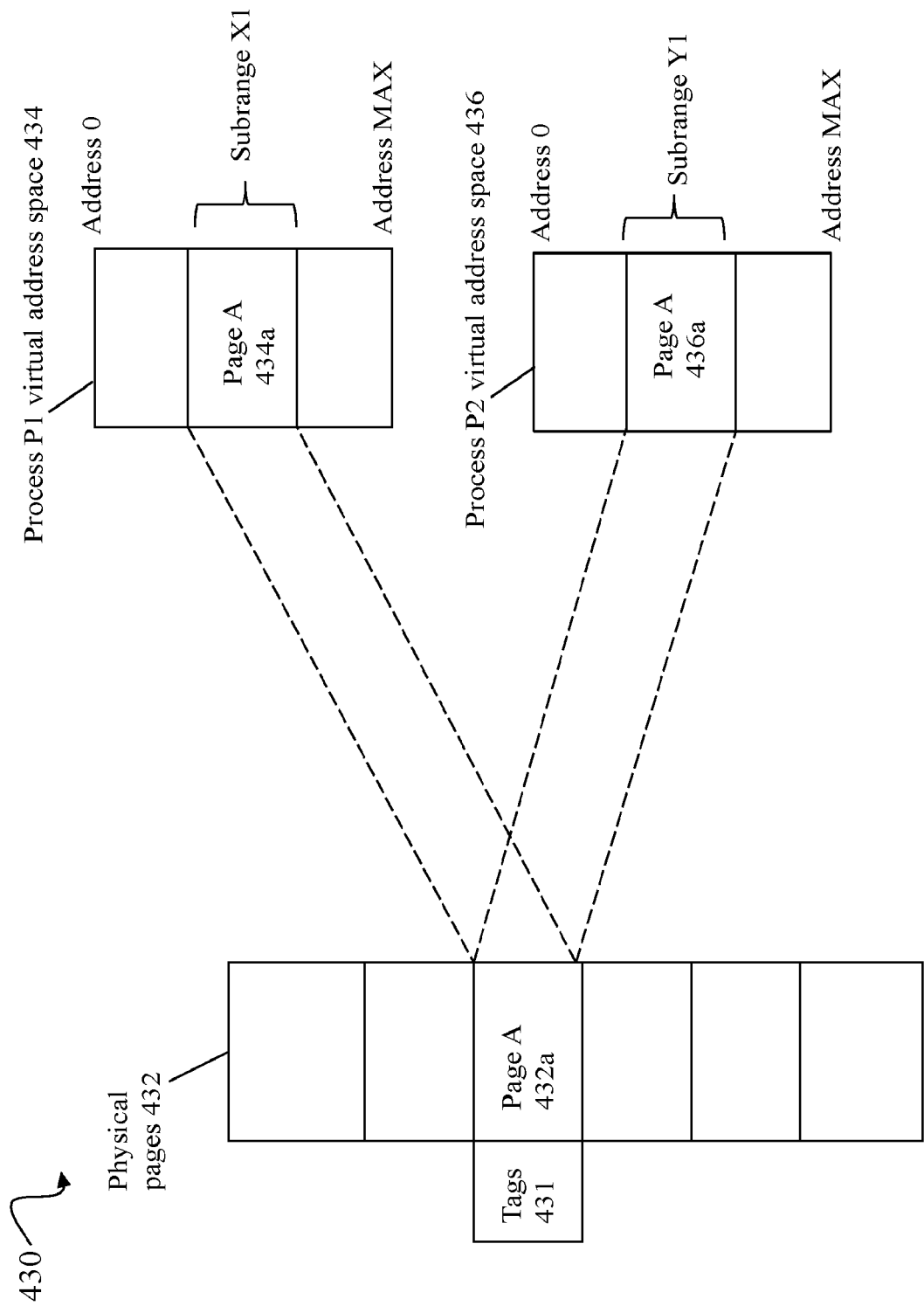


FIG. 45

500 ↗

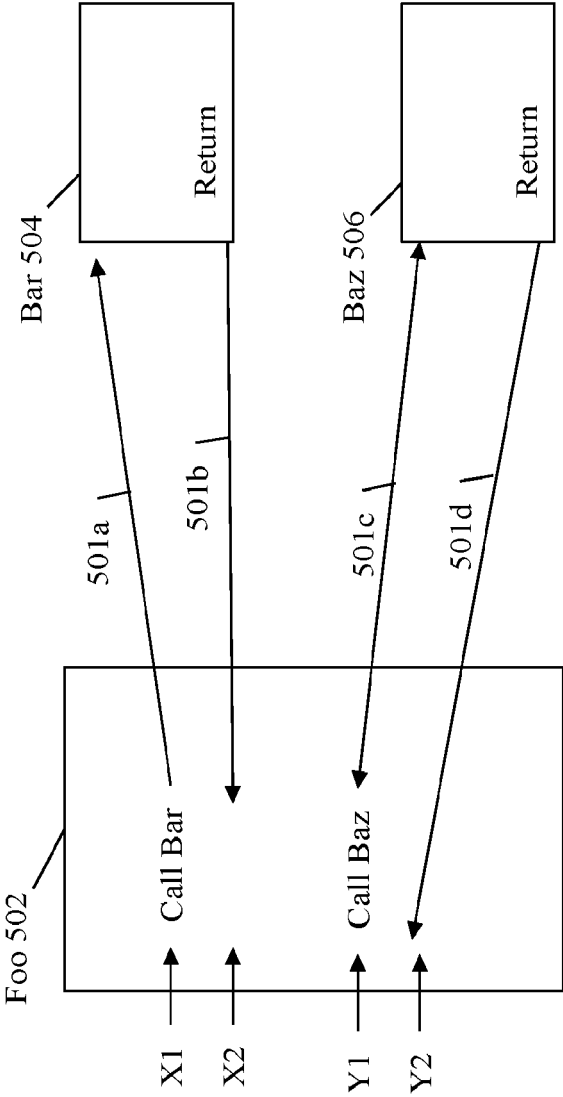


FIG. 46

520 ↗

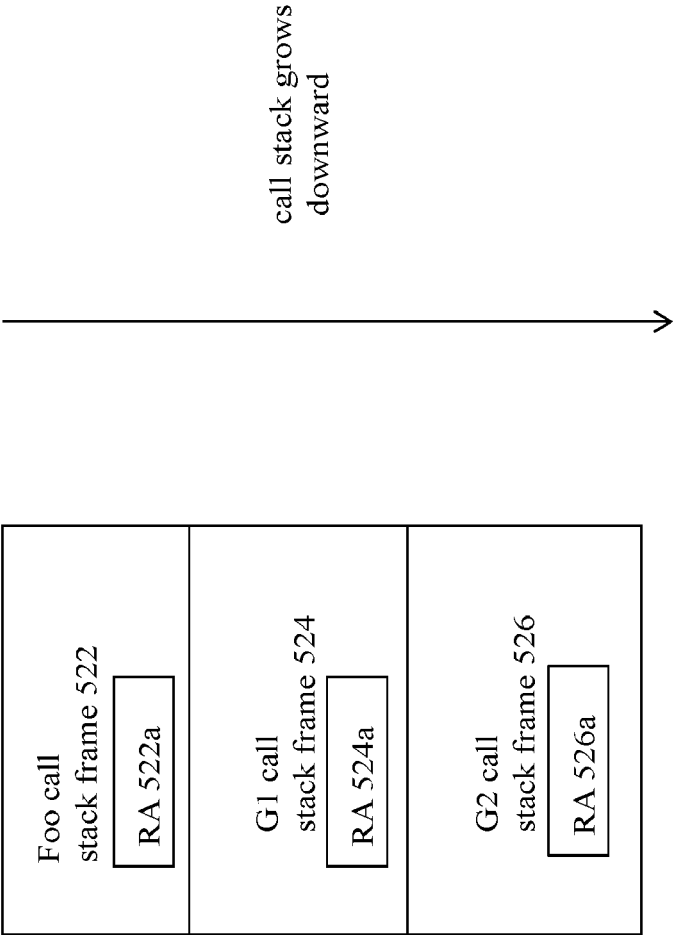


FIG. 47

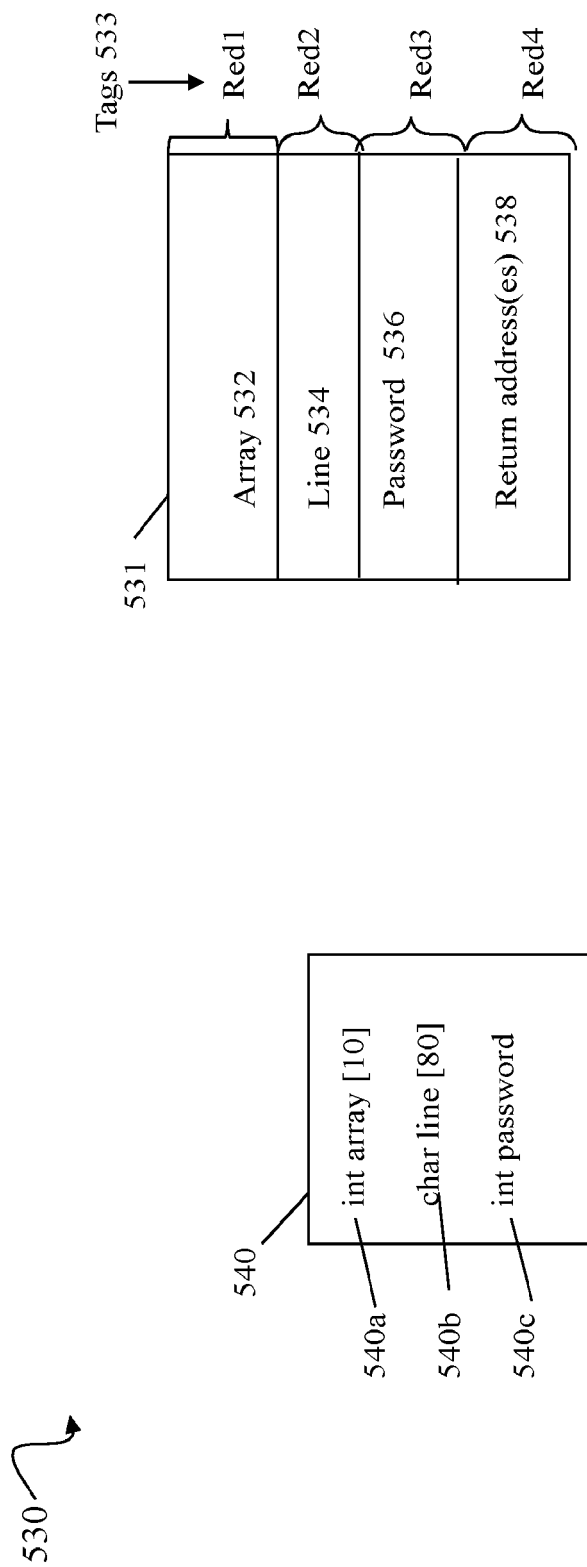


FIG. 48

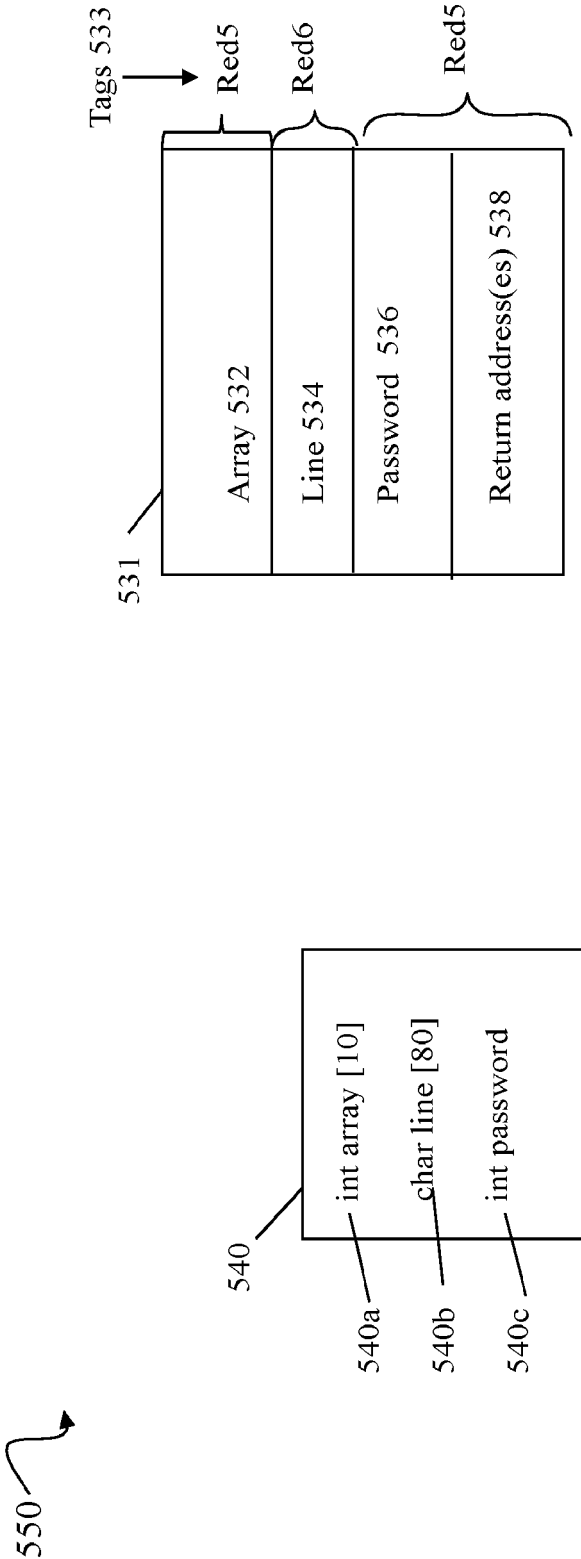


FIG. 49

560 ↗

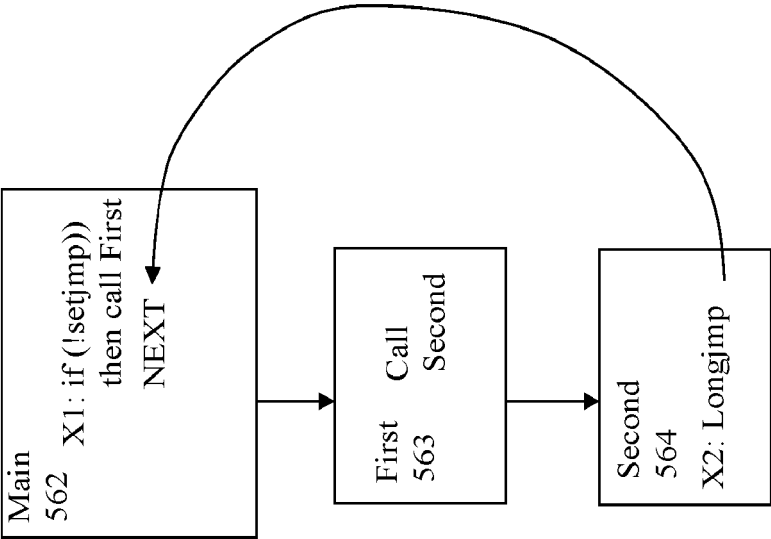


FIG. 50

570

Item/runtime behavior to prevent	Preventive Action	Mechanism
Never Return	Timeout	Separate thread
		Time/instruction limited call
Exhaust resources	Limit Resources of callee	Separate thread
		Resource limited call
Exercise unexpected Authority	Limit authority to least privilege	tag PC as authority and control capabilities visible to callee
		Limit portion of file system accessible, Limit system calls that can be made
Read items left in registers by functions called by this code	Clear non-input/non-return registers	Separate process
		Explicit register clearing
		Coloring
Read items left on stack by functions called by this code	Make called stack inaccessible	Separate process
		Separate Stack
		Capabilities
		Coloring

572a

572b

572c

572d

572e

FIG. 51

575

Item/runtime behavior to prevent	Preventive Action	Mechanism
572f Write over items on stack prefix	Make stack prefix inaccessible or otherwise unwritable	Separate process
		Separate stack
		Capabilities
		Coloring
572g Read items in stack prefix	Make stack prefix inaccessible	Separate process
		Separate stack
		Capabilities
		Coloring
572h Redirect control flow in Prefix (e.g., overwrite return pointer)	Protect return address	Separate process
		Separate stack
		Coloring- tag stack location storing return address
		Capability - tag return pointer by access authority

FIG. 52

580	Item/runtime behavior to prevent	Preventive Action	Mechanism
581a	Write over unintended items in current frame	Maintain object integrity	capabilities by object color by object
581b	Read items in current frame	Maintain object integrity	capabilities by color color by object
581c	Write over unintended items in predecessor frame	Isolate frames	capabilities by frame coloring by frame
581d	Read items in predecessor frame	Isolate frames	capabilities by frame coloring by frame
581e	Read items left on stack by functions called by this code	Make call stack inaccessible	Separate process Separate stack Capabilities Coloring
581f	Modify return pointer	Protect return pointer	Coloring- tag stack location storing return address Capability - tag return pointer by access authority

FIG. 53

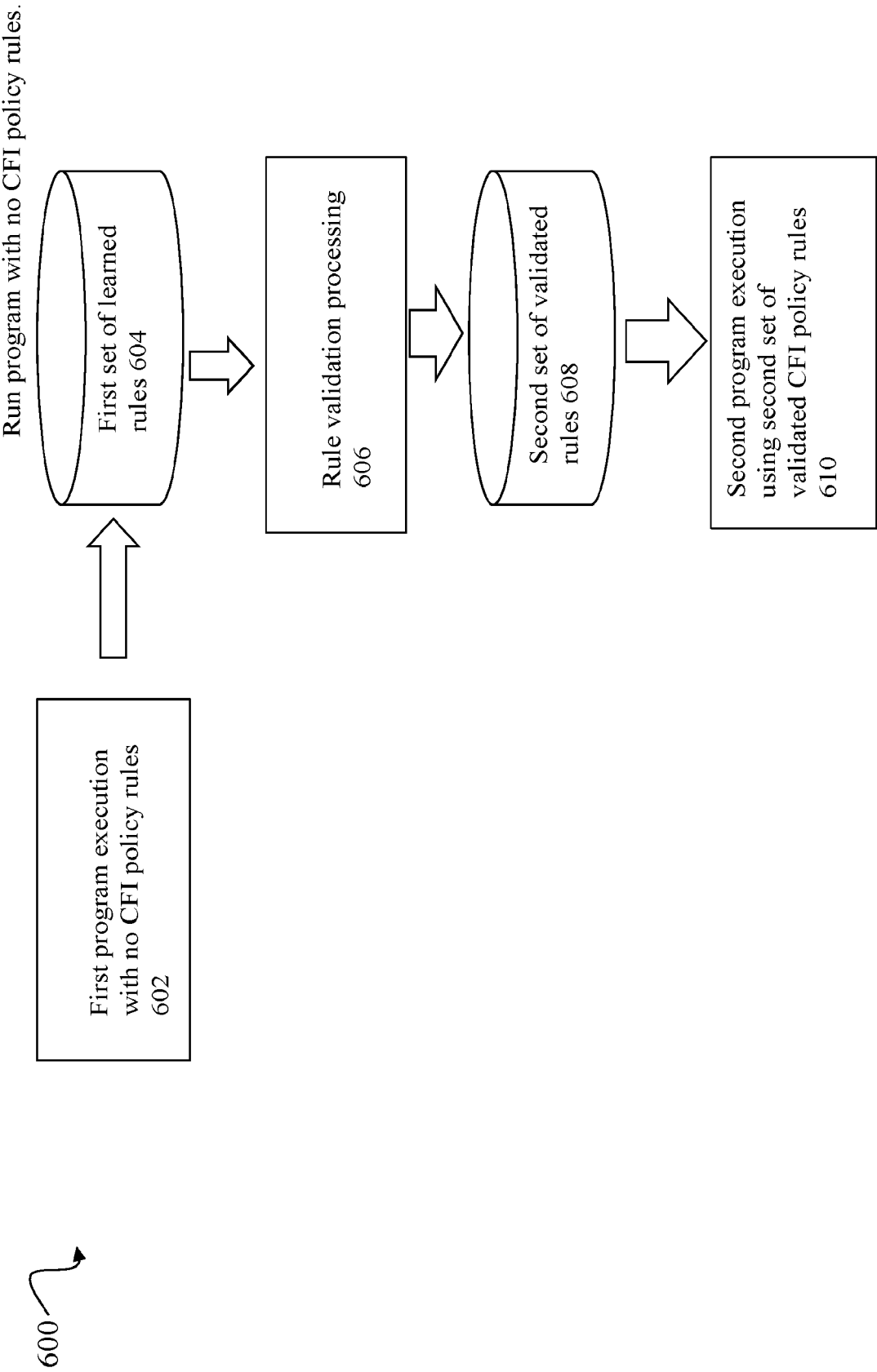


FIG. 54

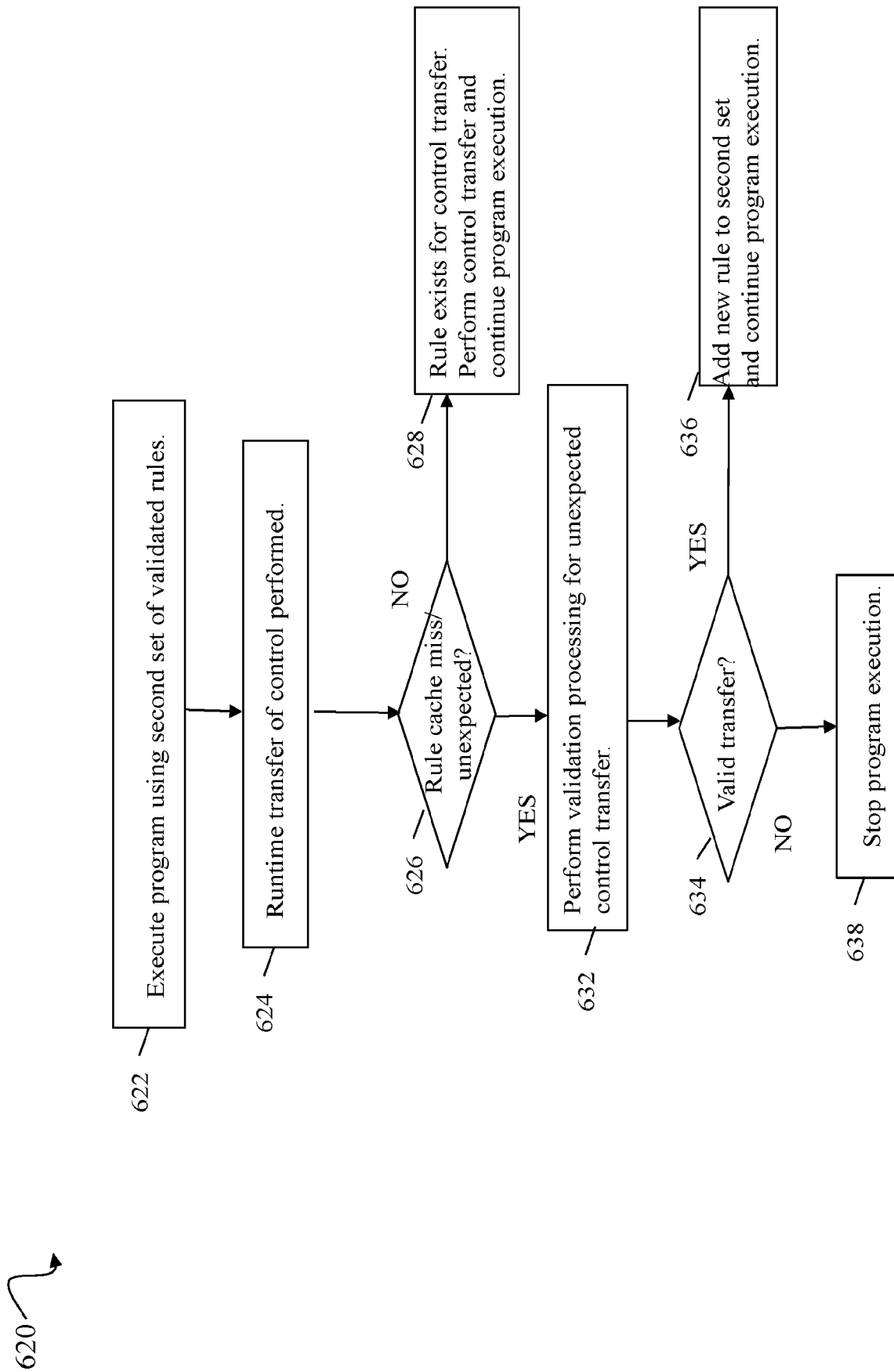


FIG. 55

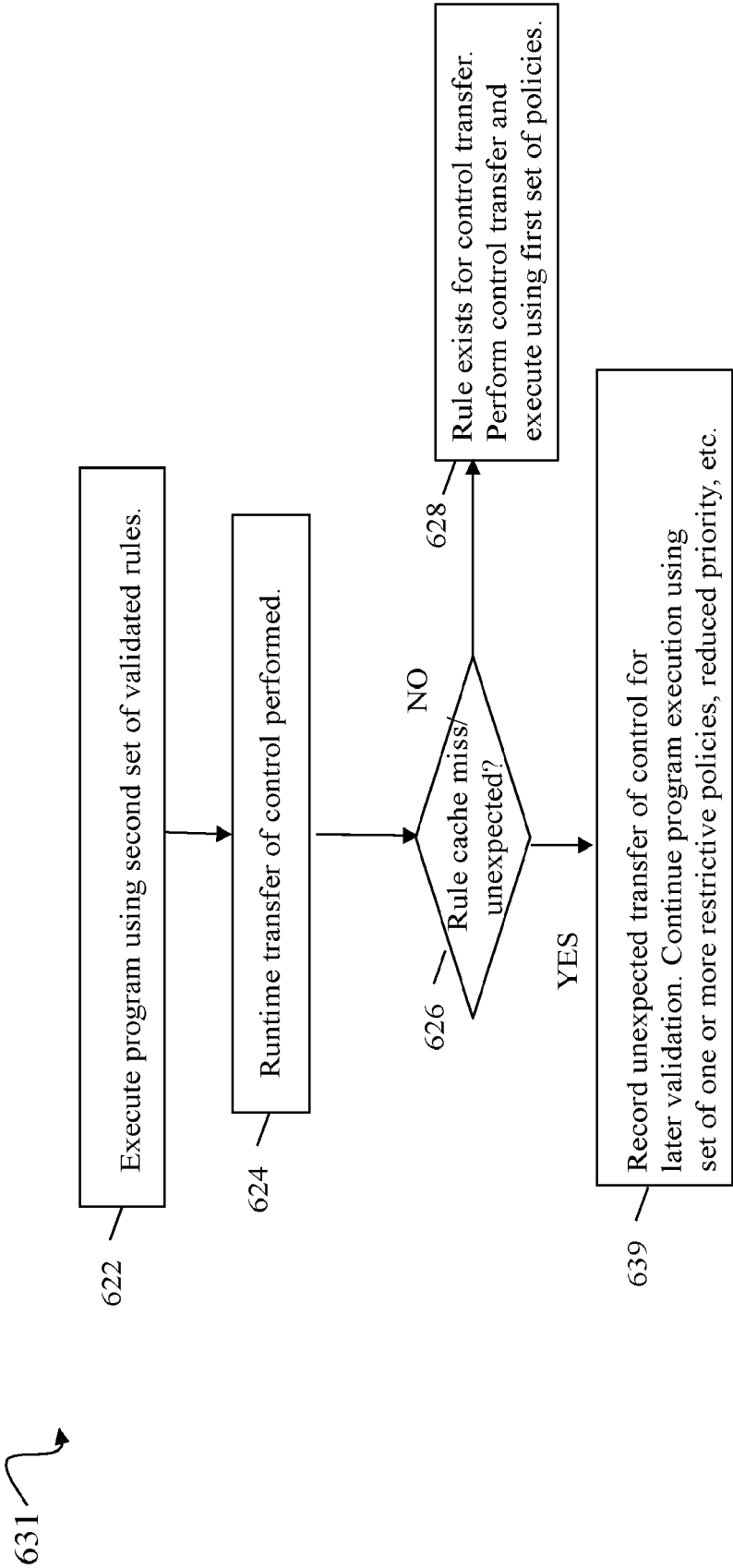


FIG. 56

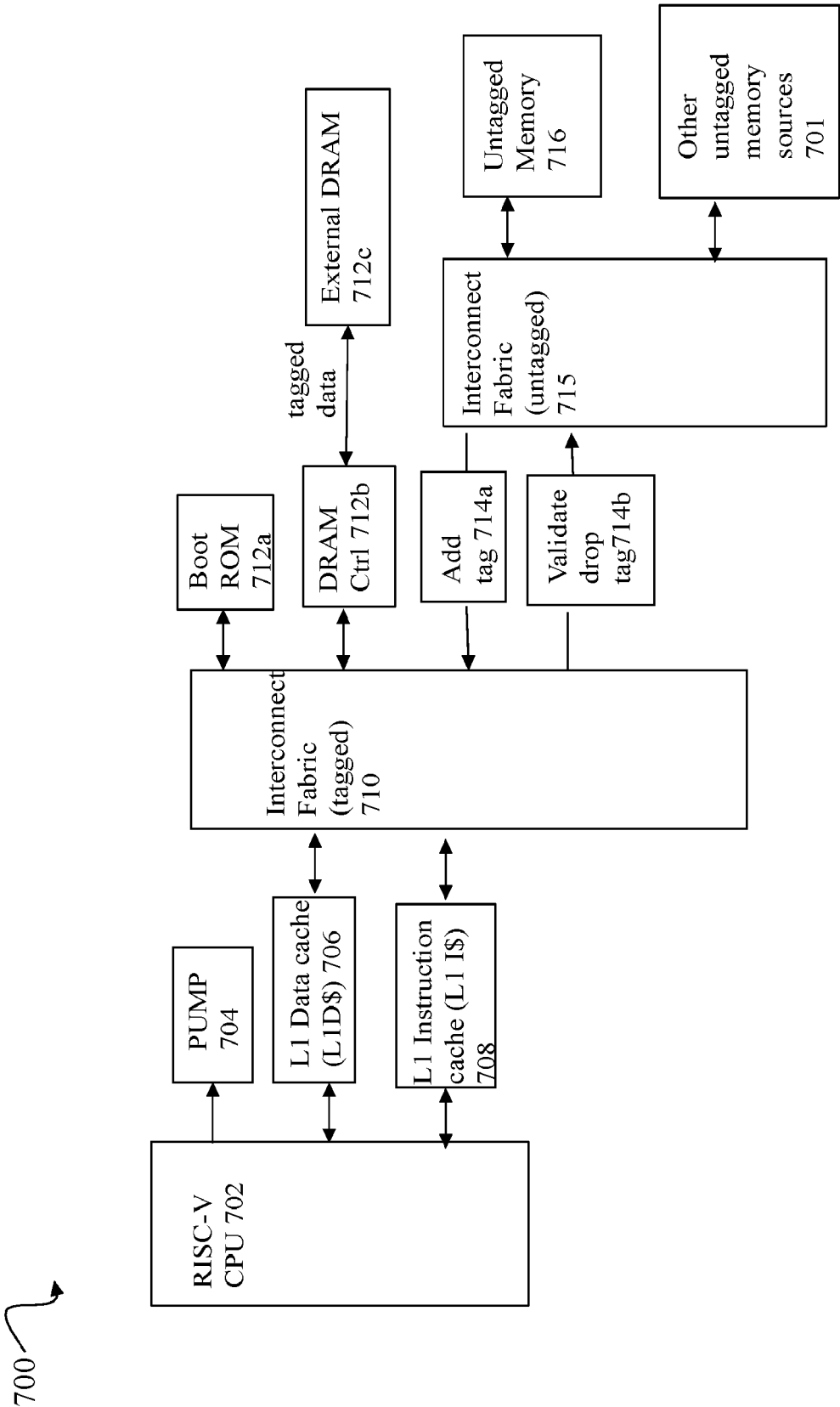


FIG. 57

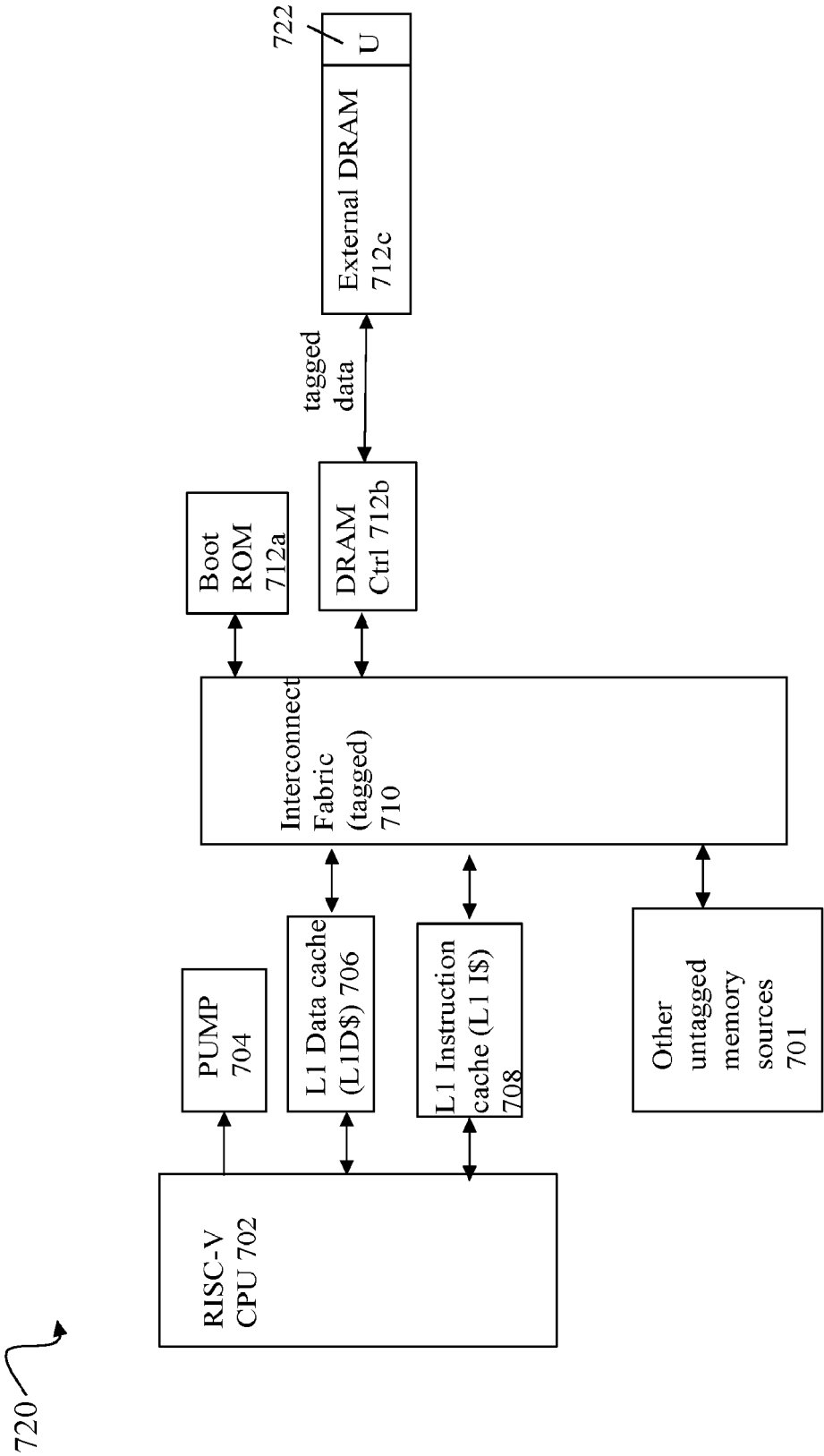


FIG. 58

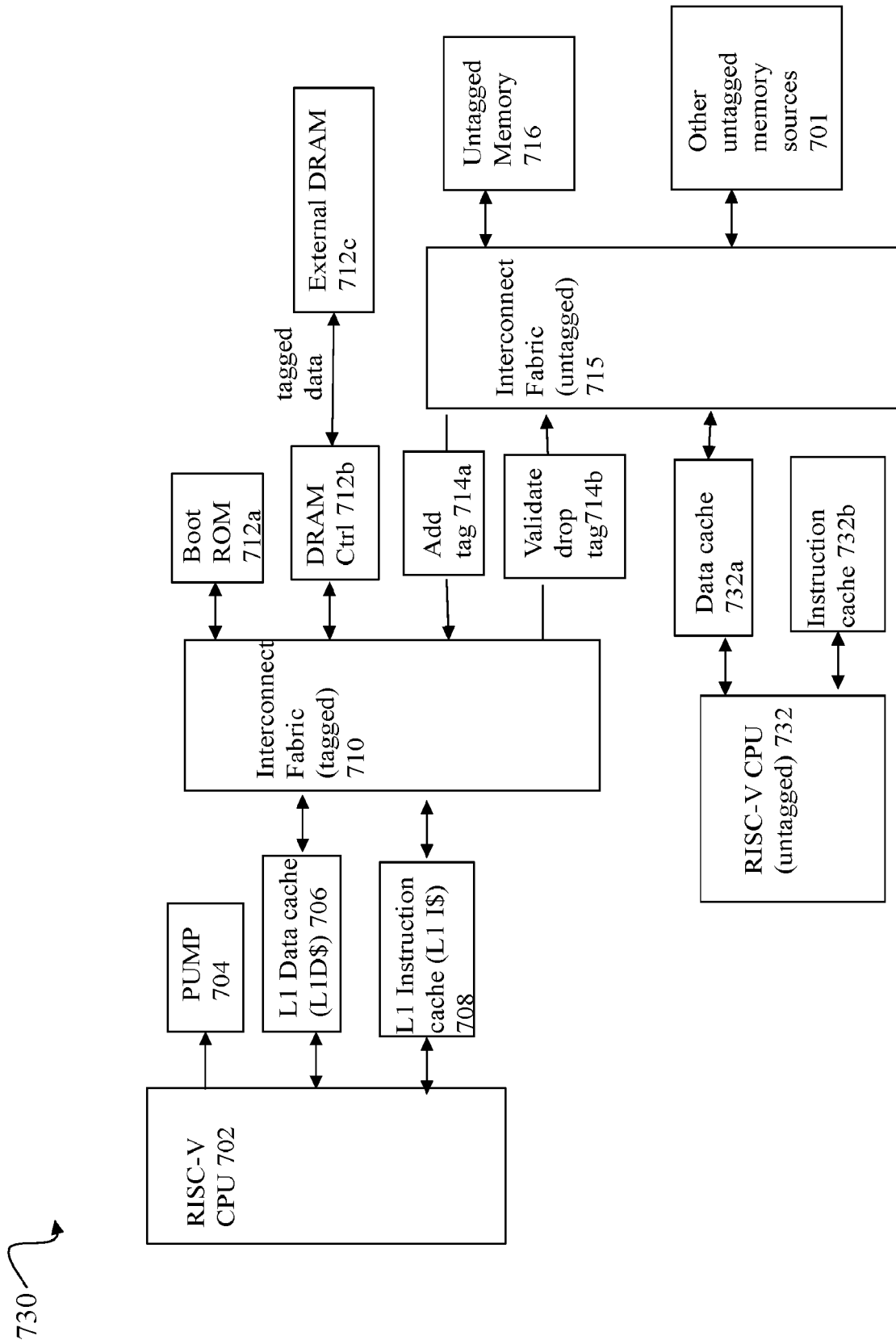


FIG. 59

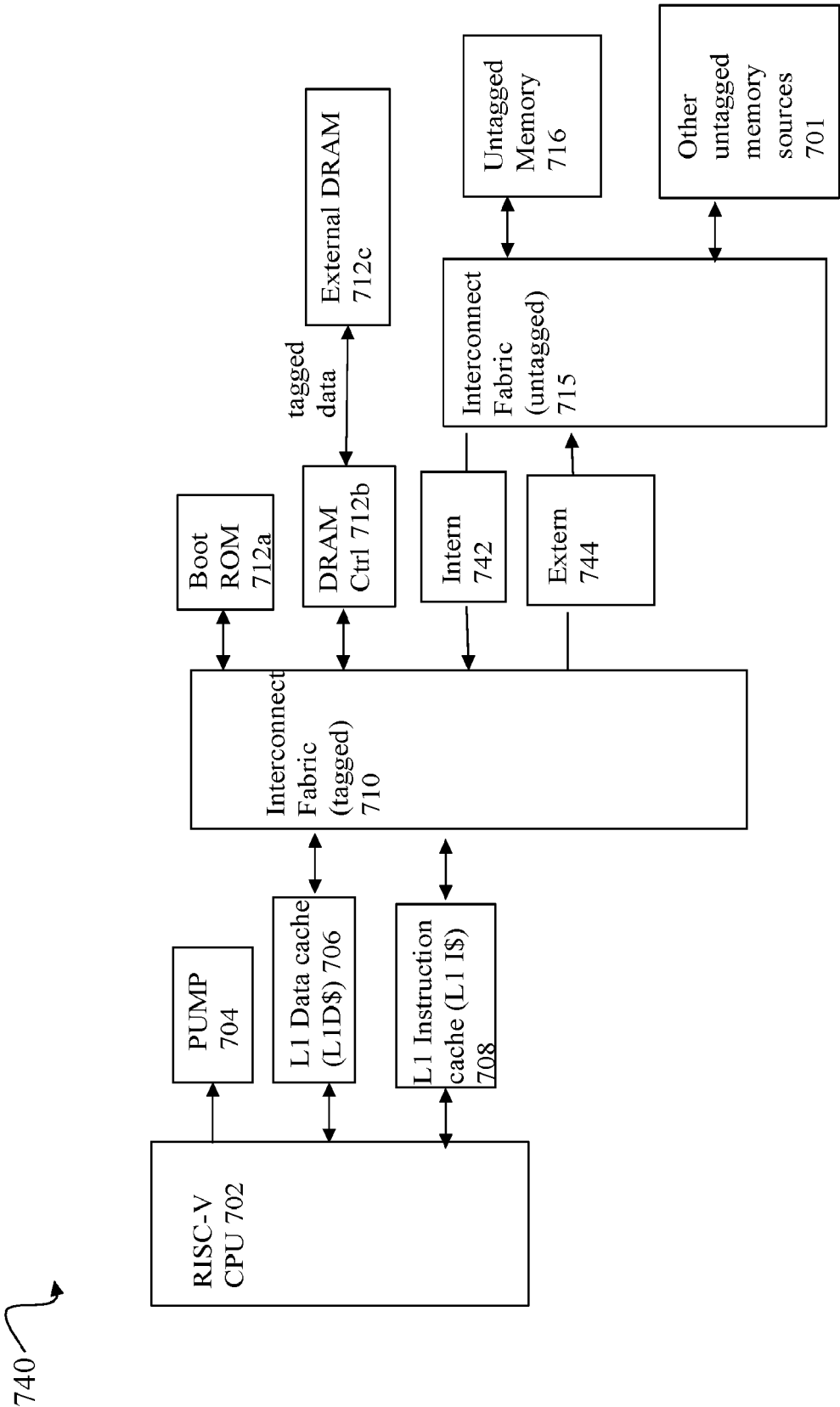


FIG. 60

800 ↗

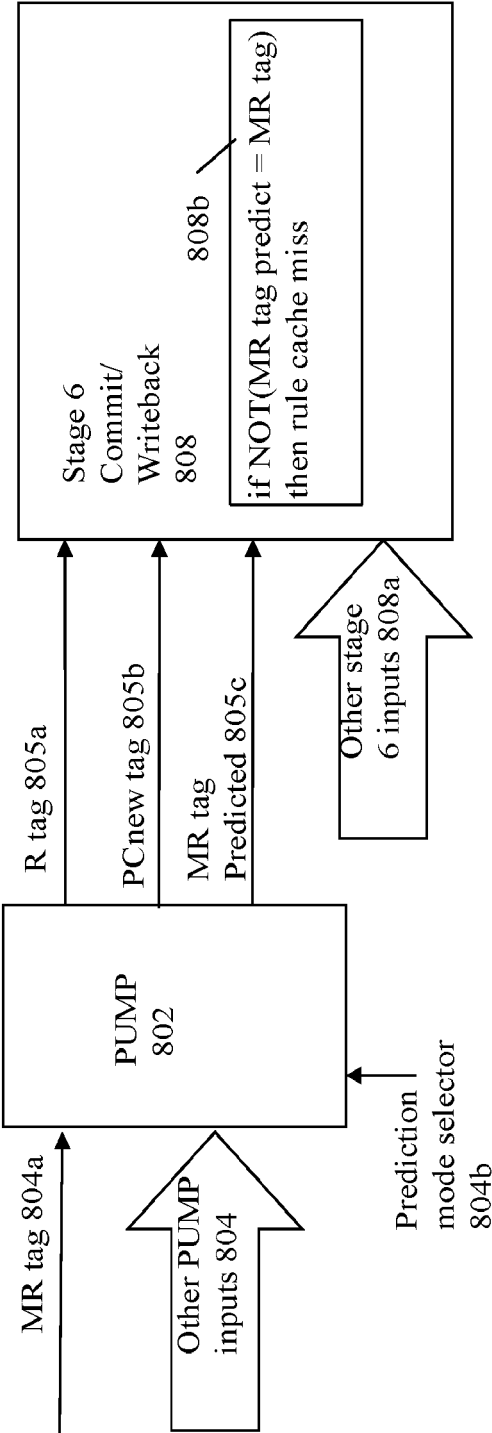


FIG. 61

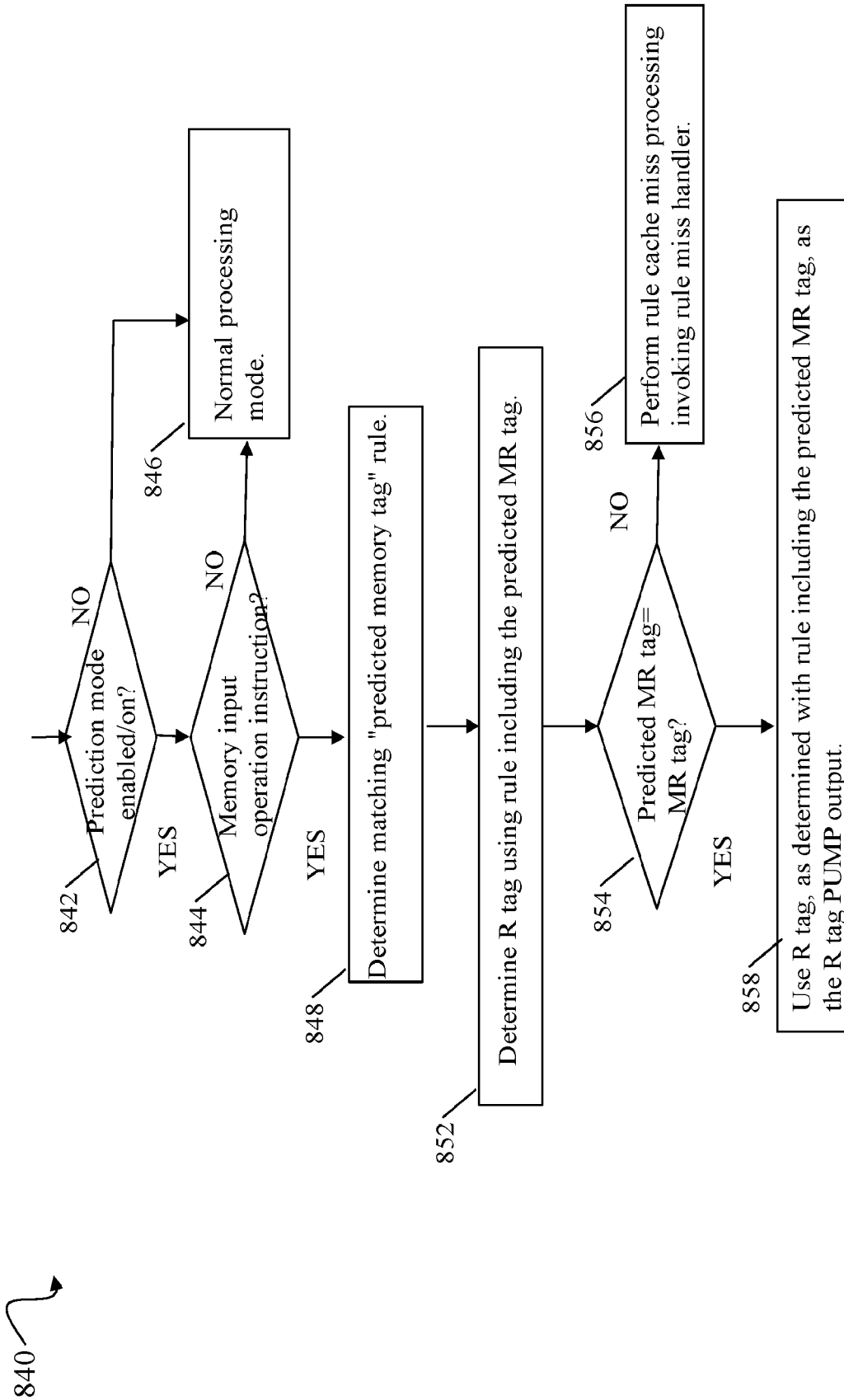


FIG. 62

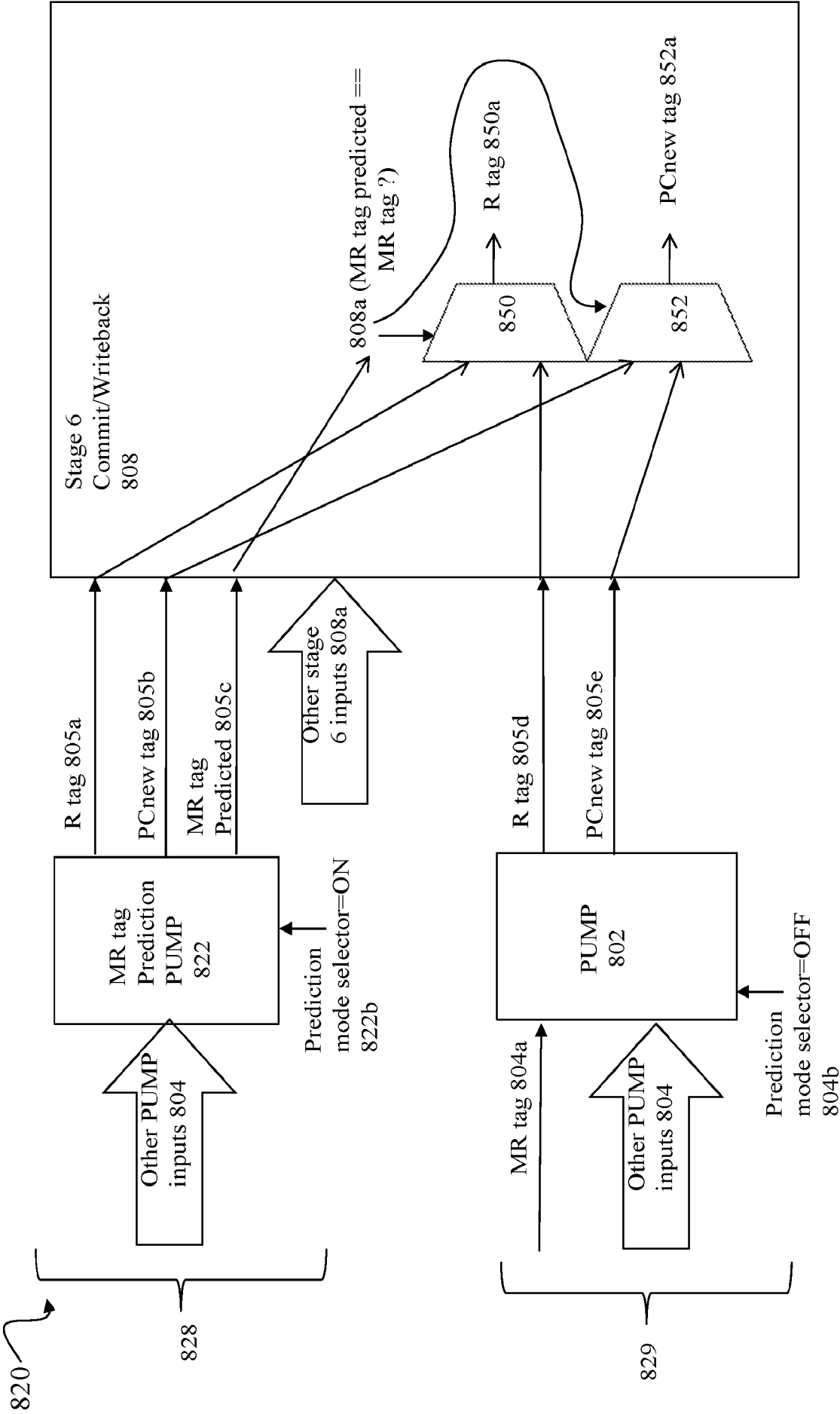


FIG. 63

70/90

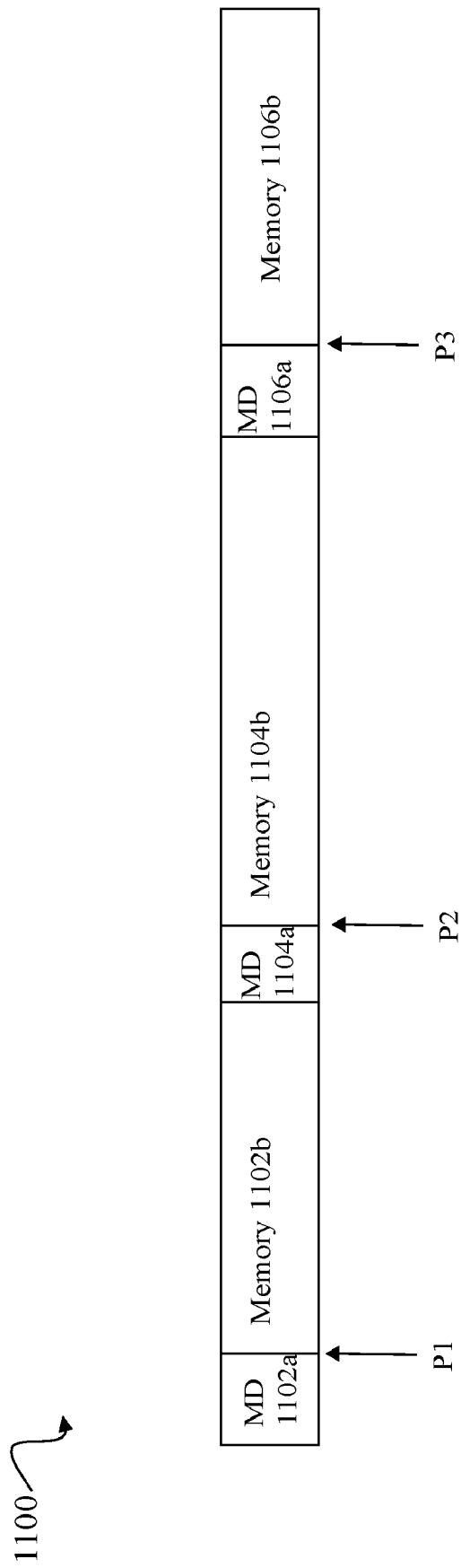


FIG. 64

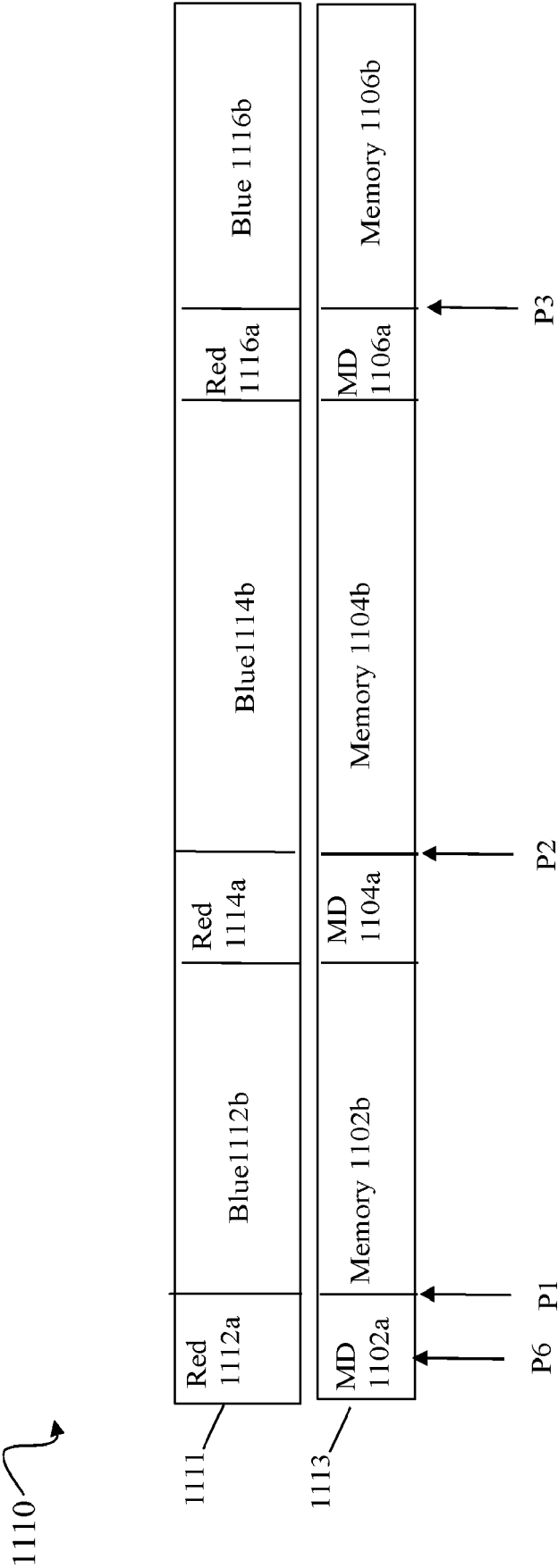


FIG. 65

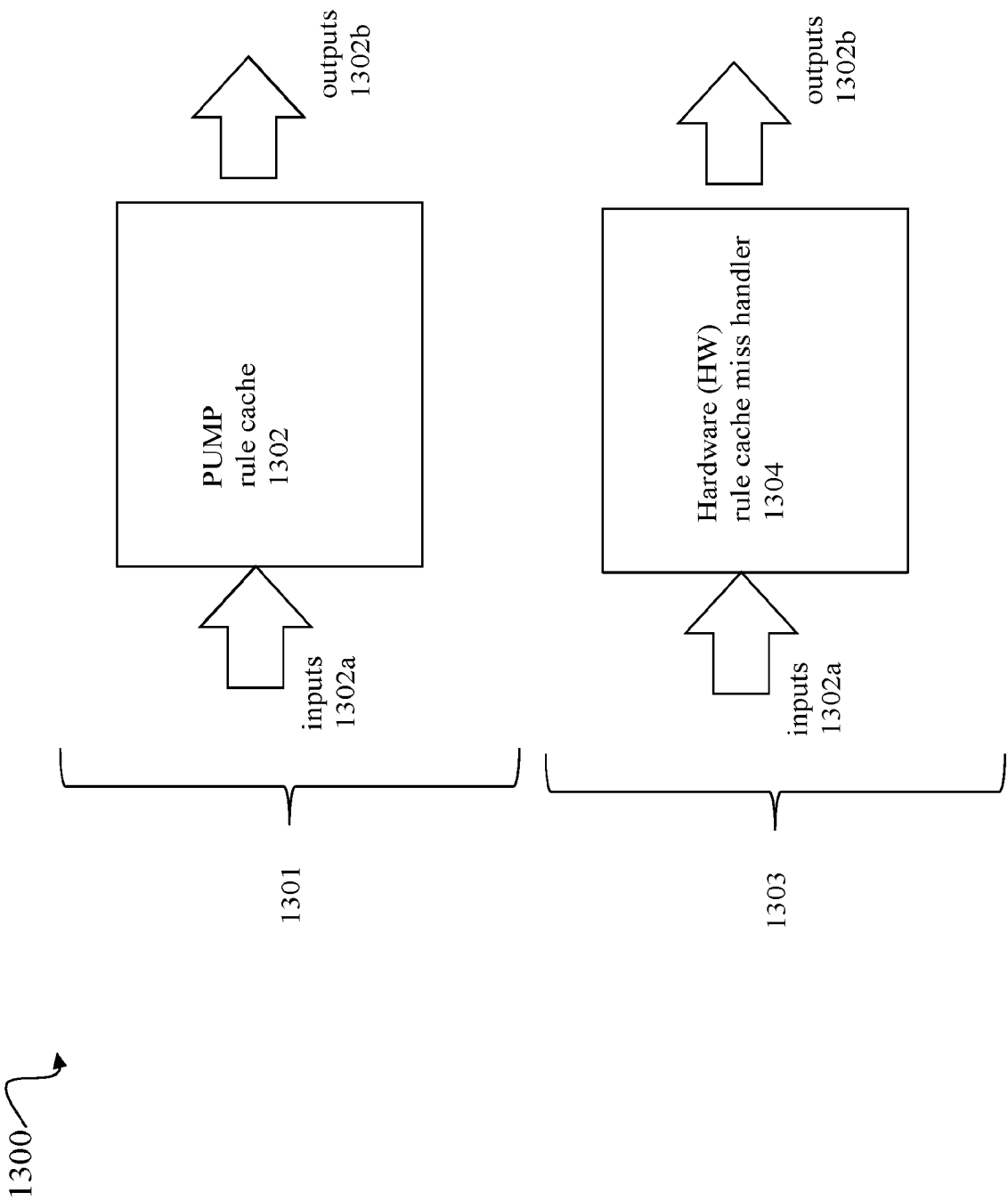


FIG. 66

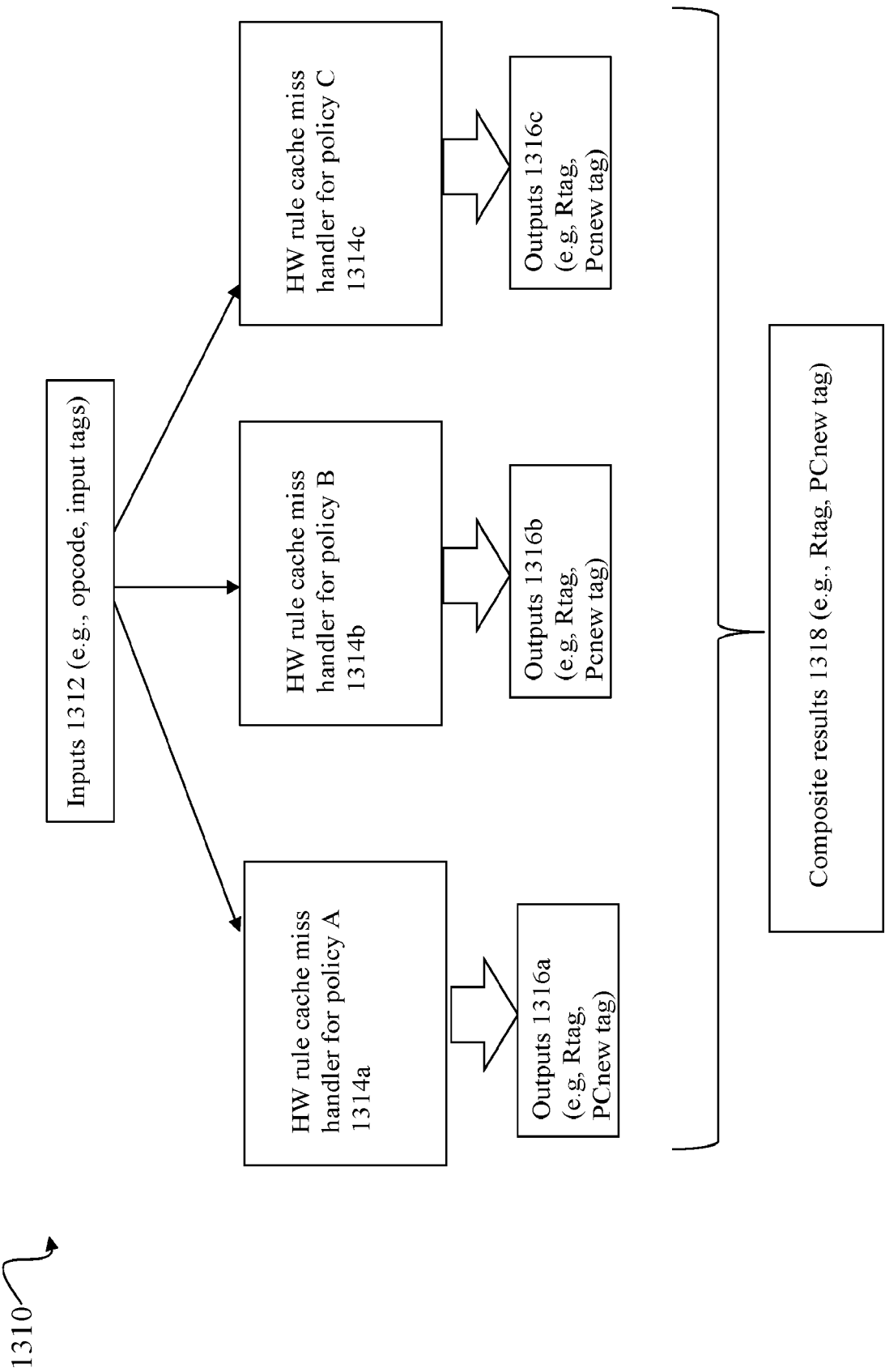


FIG. 67

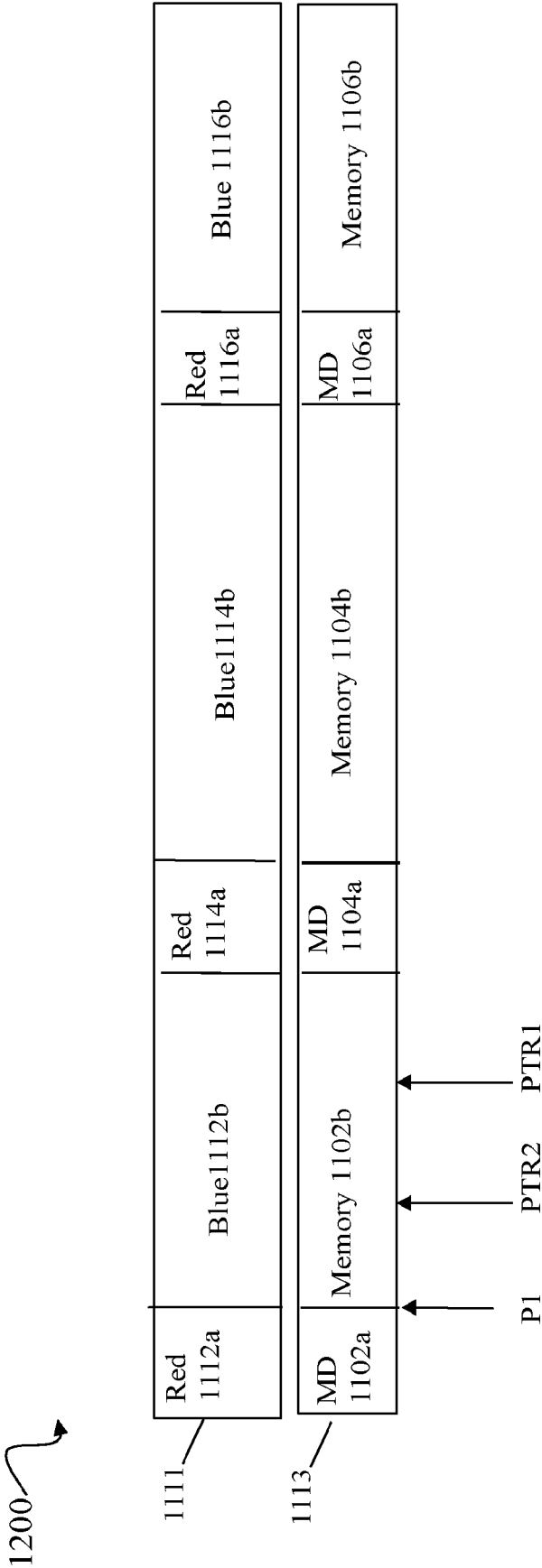


FIG. 68

1210

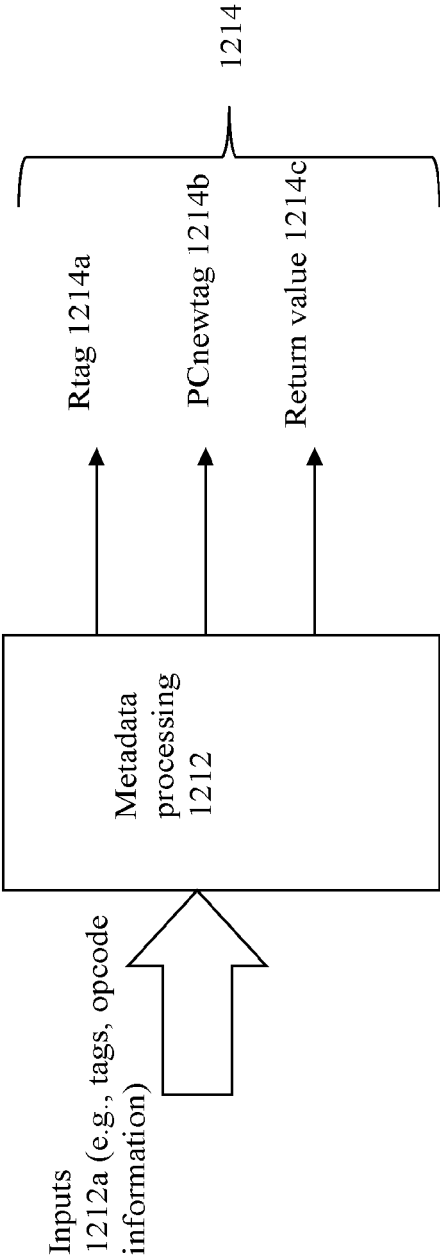


FIG. 69

1220

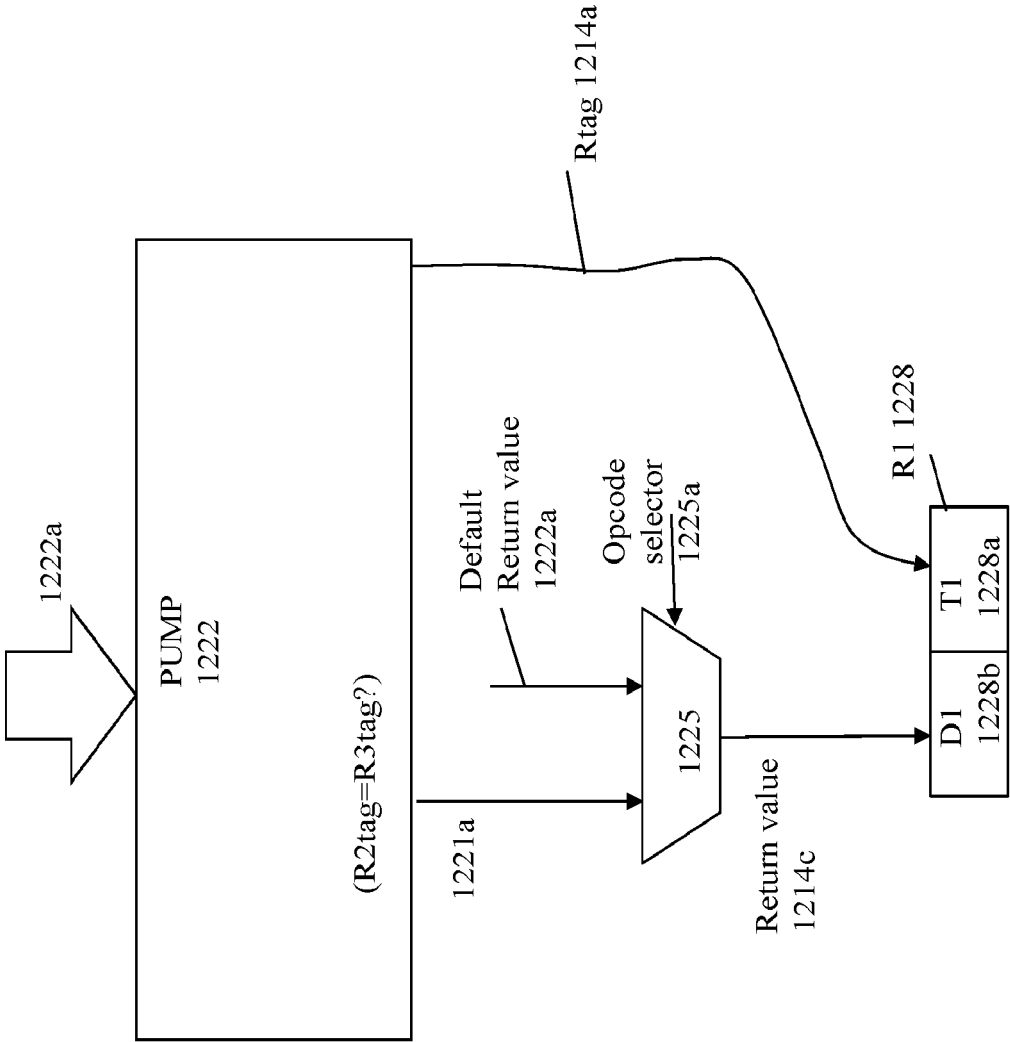


FIG. 70

77/90

1400 ↗

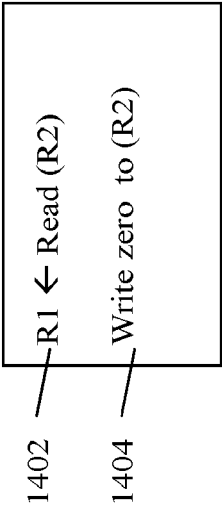


FIG. 71

78/90

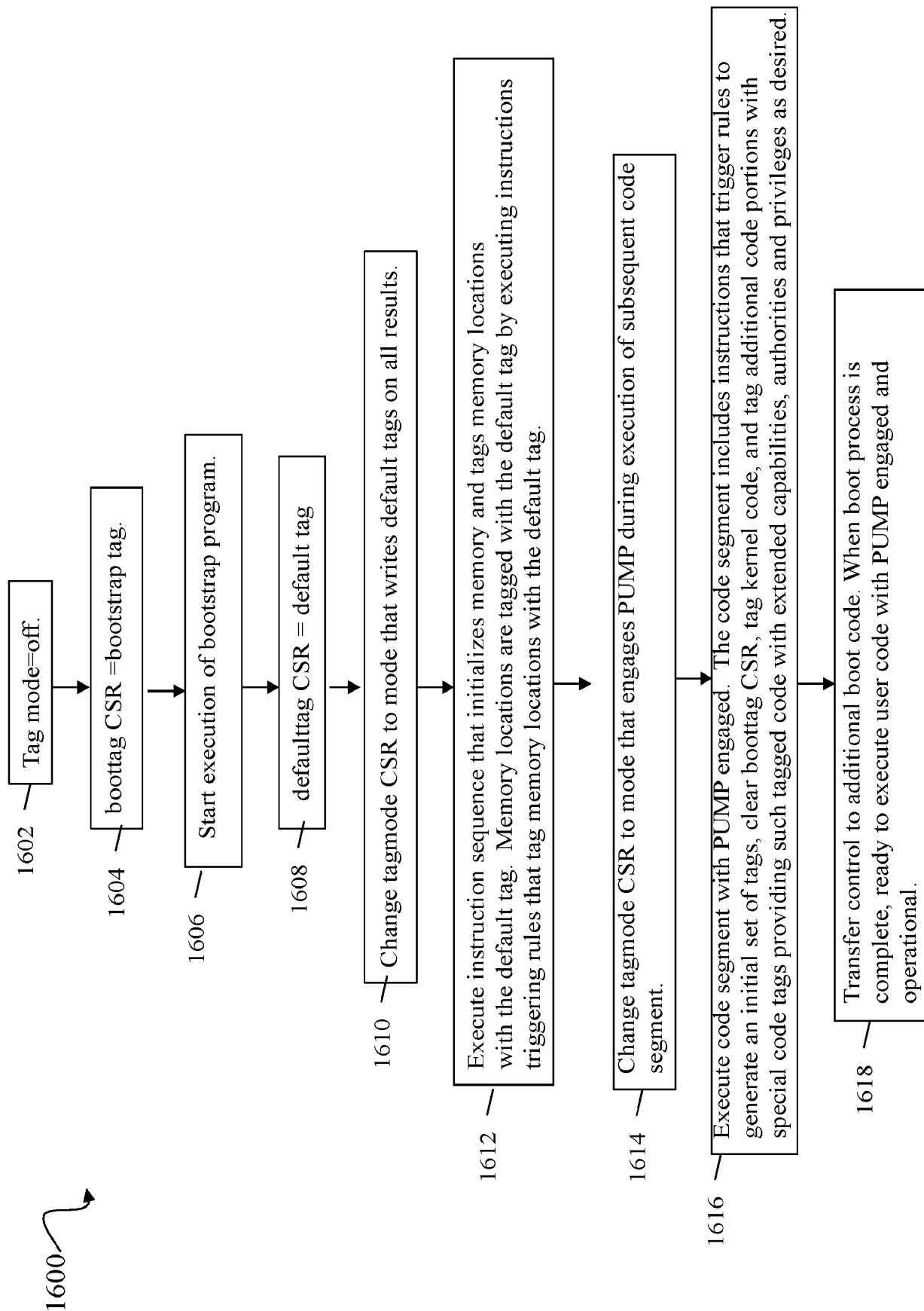


FIG. 72

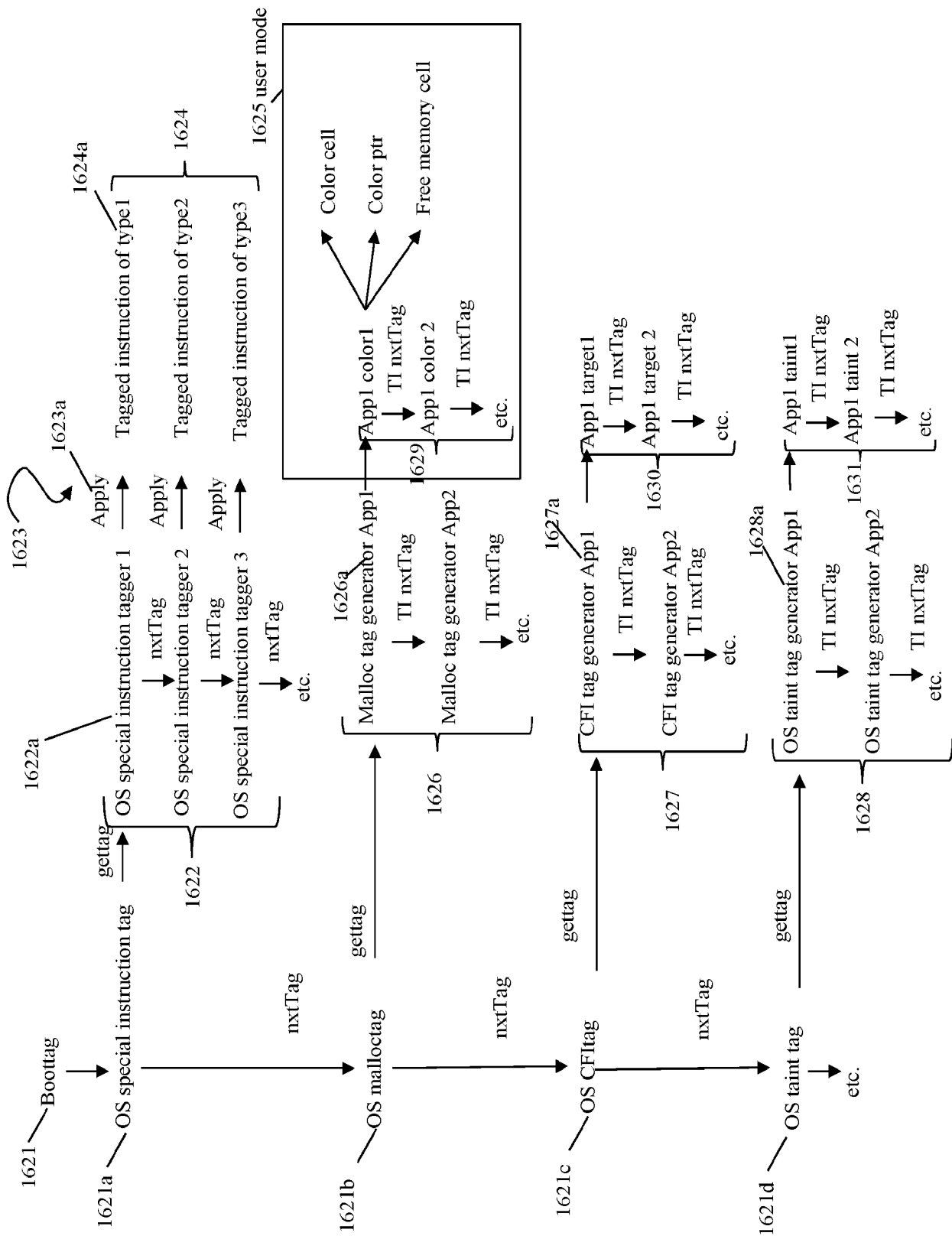


FIG. 73

80/90

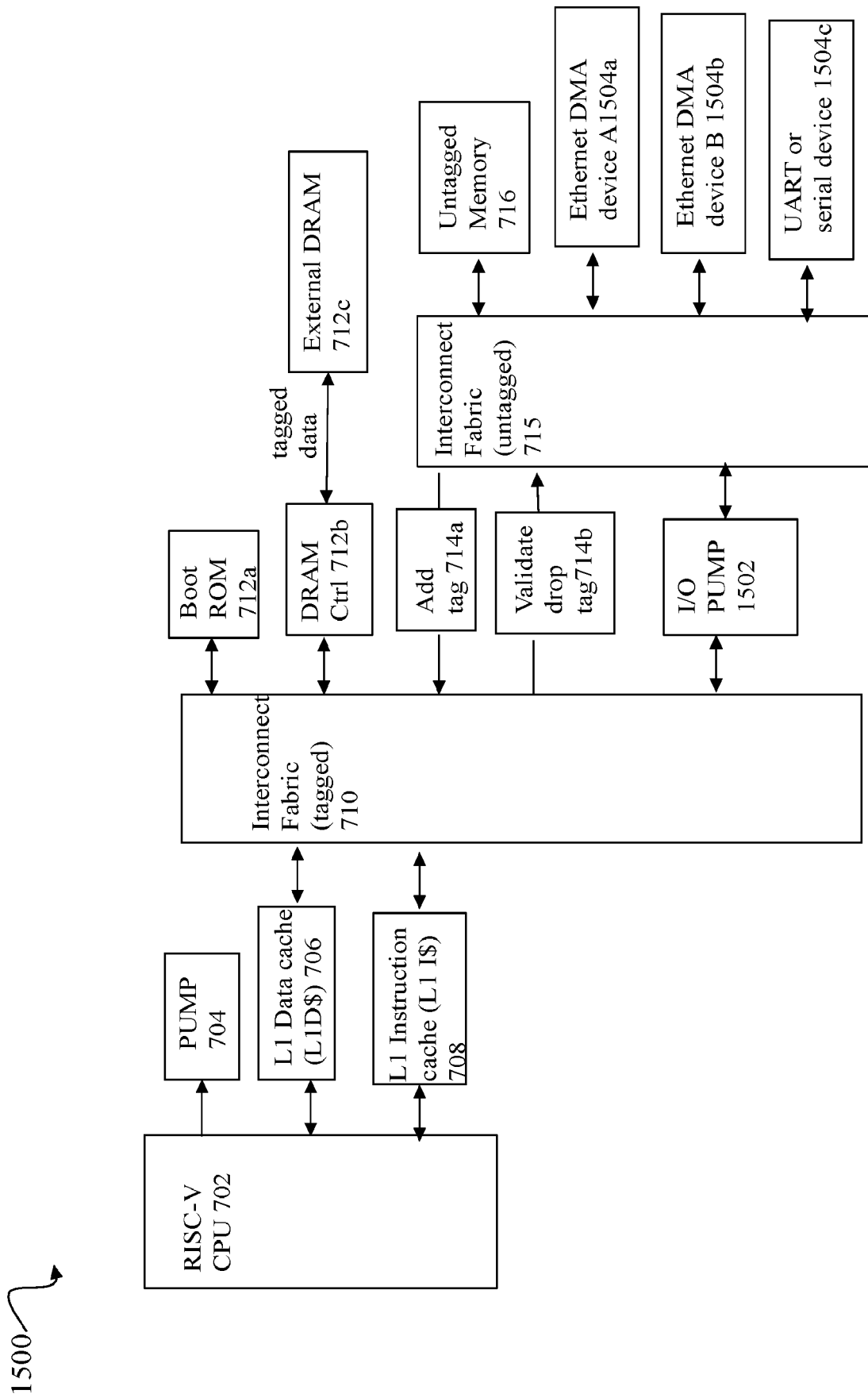


FIG. 74

81/90

1520

Address	Name	Description
0x00	transaction id	Write to this address increments the transaction (for prefetch); Reads return the current transaction id
0x01	opgrp	Opgroup for rule miss
0x02	byteenable	Byte Enables for rule miss
0x03	pctag	PC tag for rule miss
0x04	citag	CI (current instruction) tag for rule miss
0x07	mtag	memory tag for rule miss
0x08	newpctag	Tag to place on PC on completion of instruction
0x09	rtag	Tag to place on memory result from instruction
0x0A	commit	when the write value matches the current transaction id, triggers write of rule to the IOPUMP
0x0E	status	enable/fault status of IOPUMP
0x0F	flush	Write to this address triggers IOPUMP flush

FIG. 75

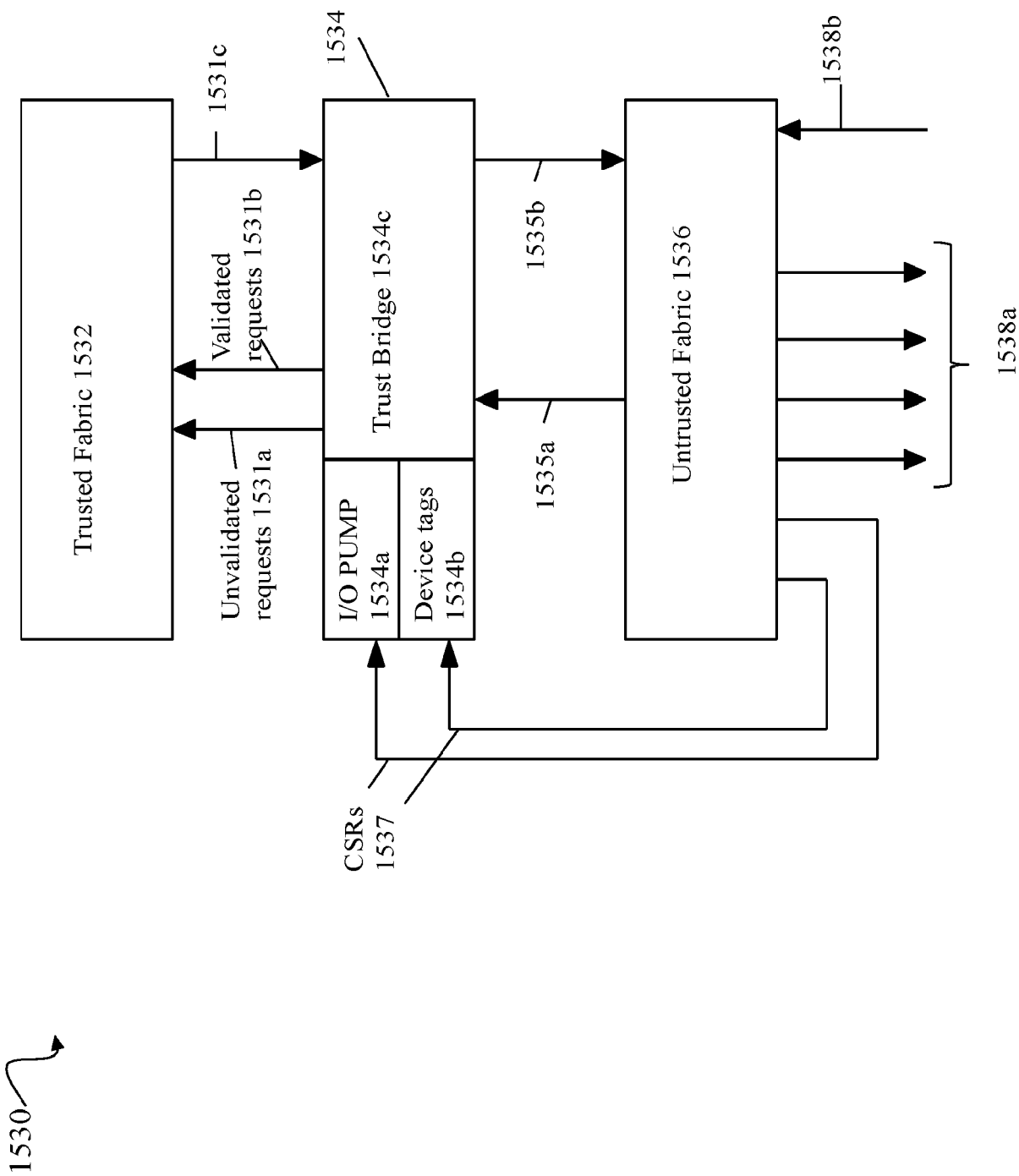
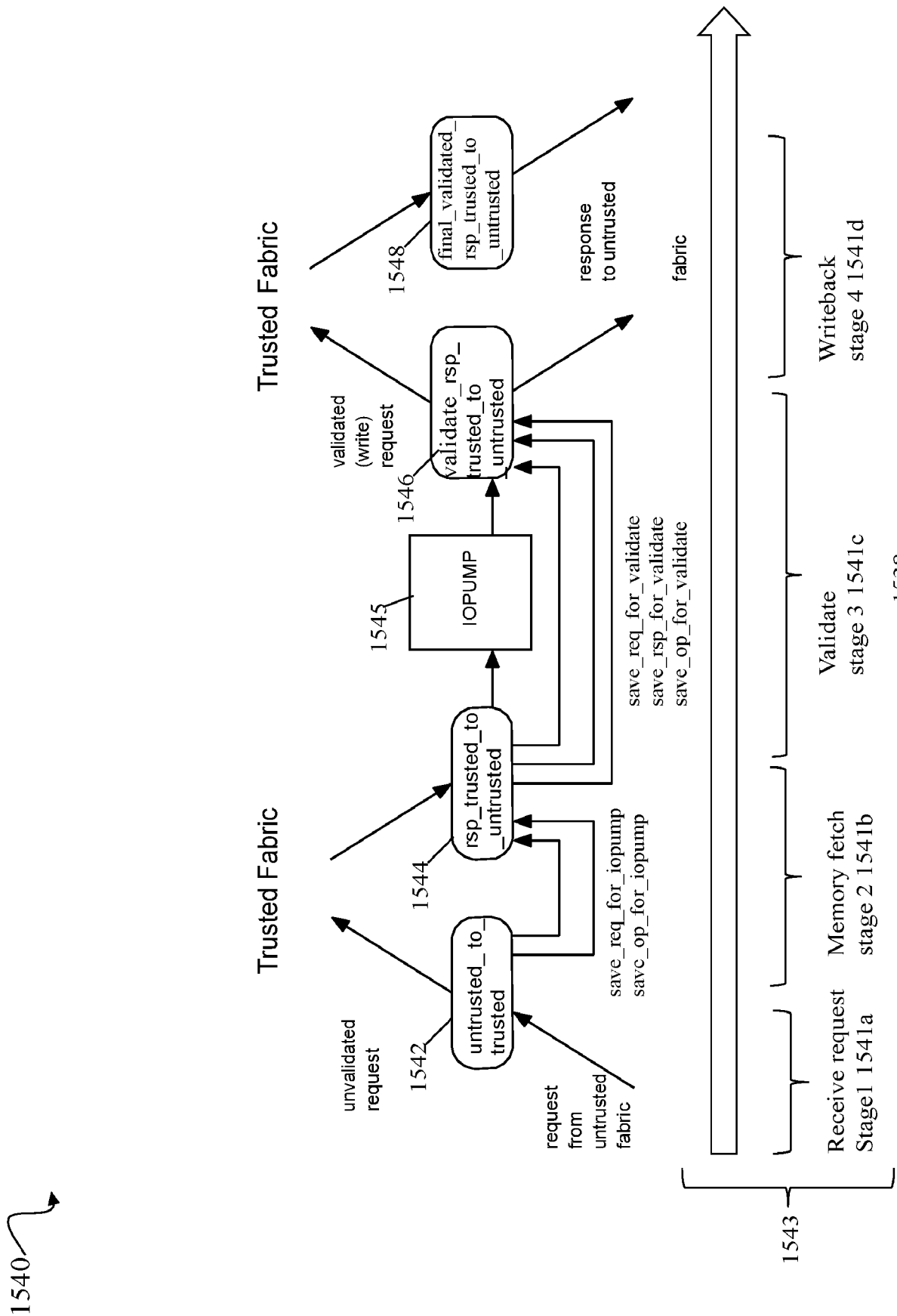


FIG. 76

83/90



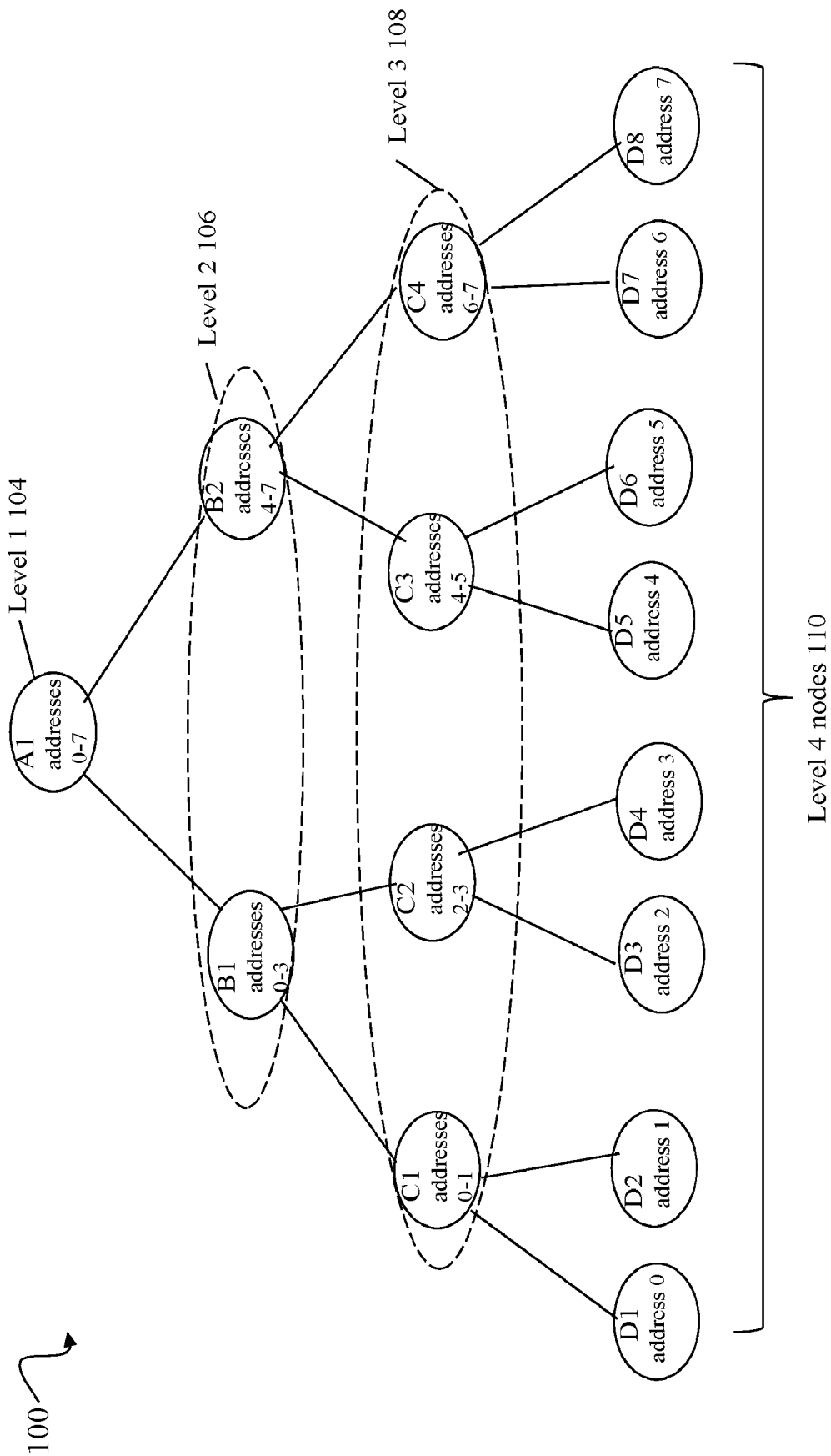


FIG. 78

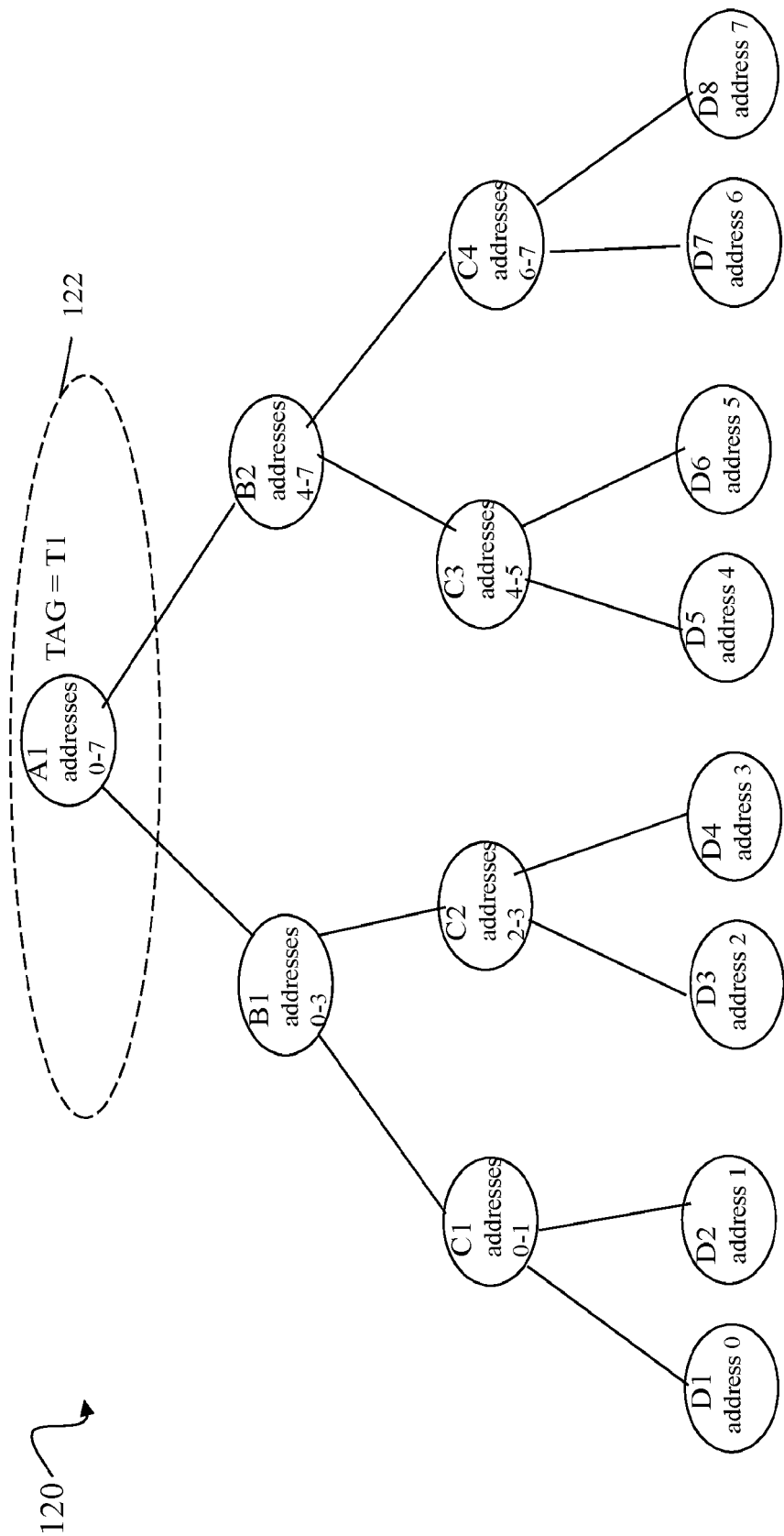


FIG. 79

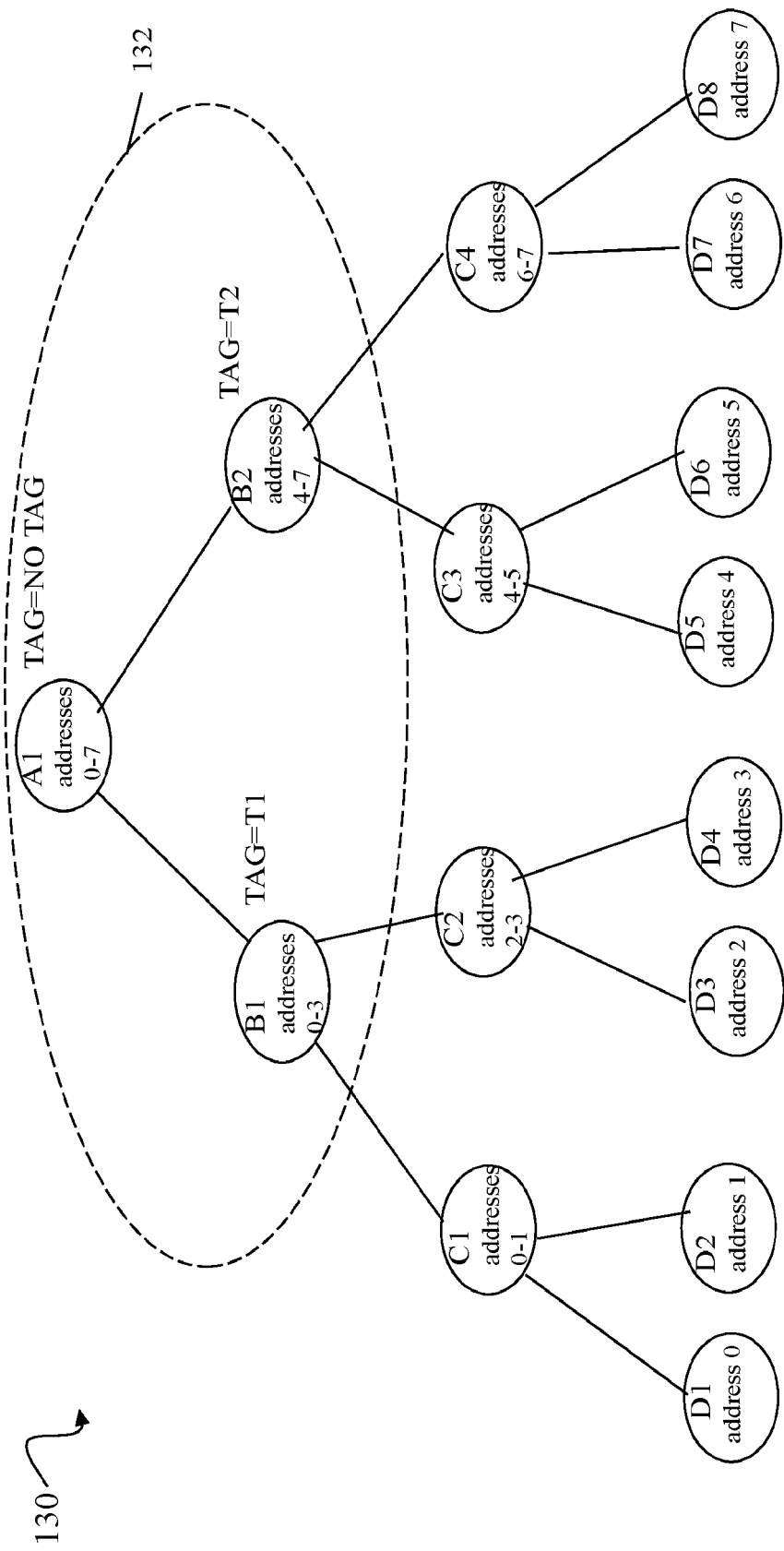


FIG. 80

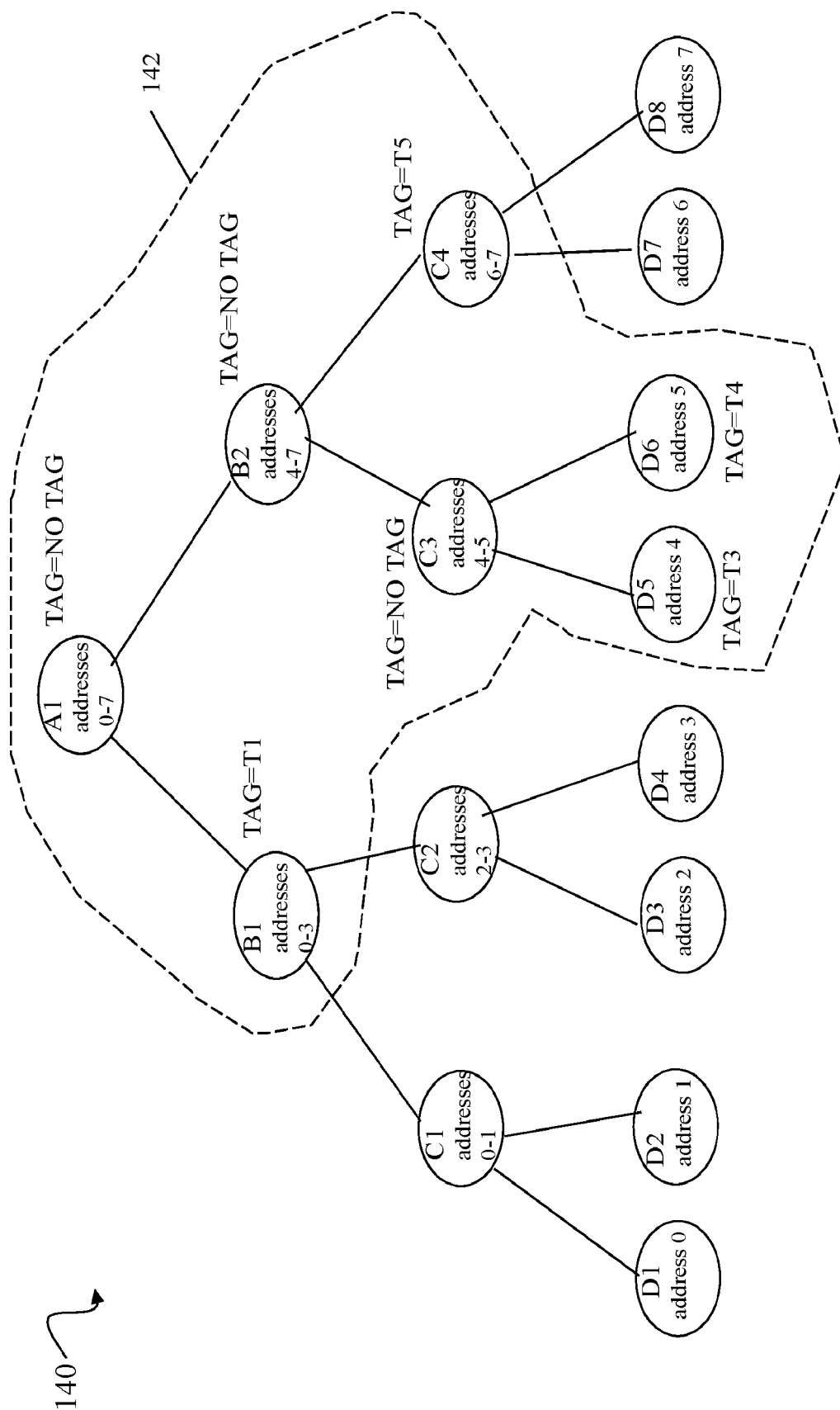


FIG. 81

150 ↗

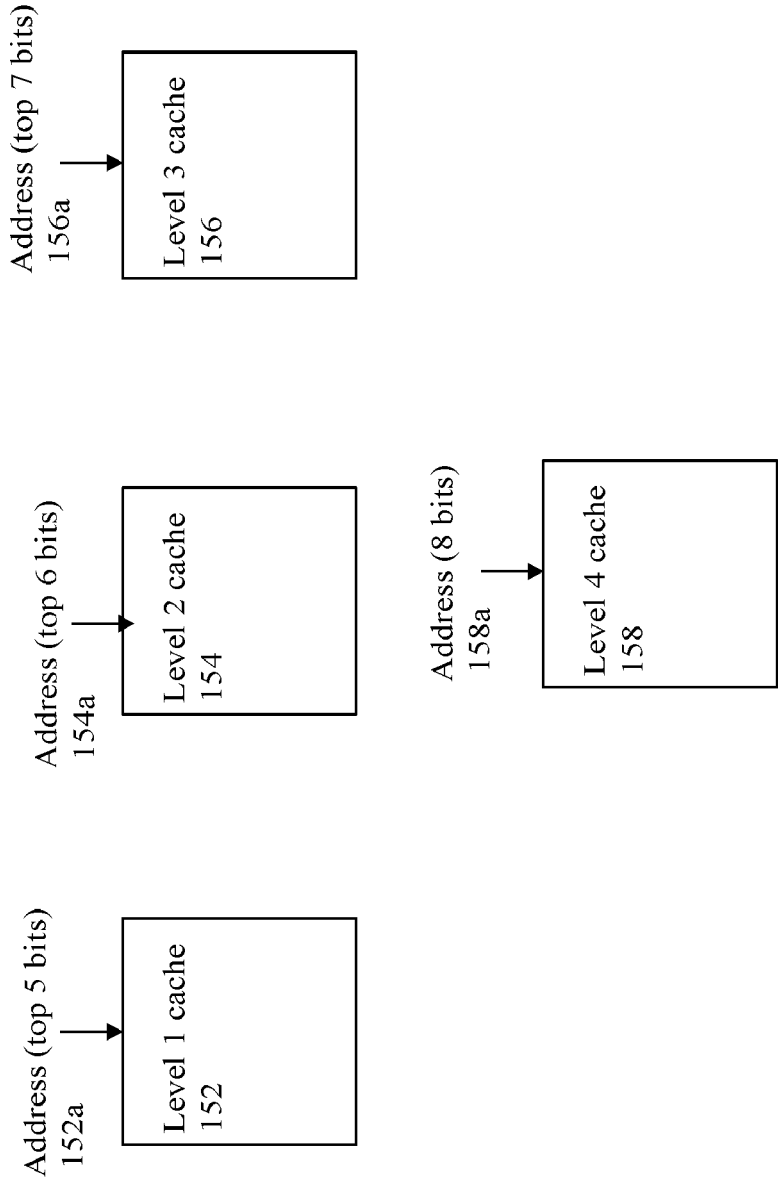


FIG. 82

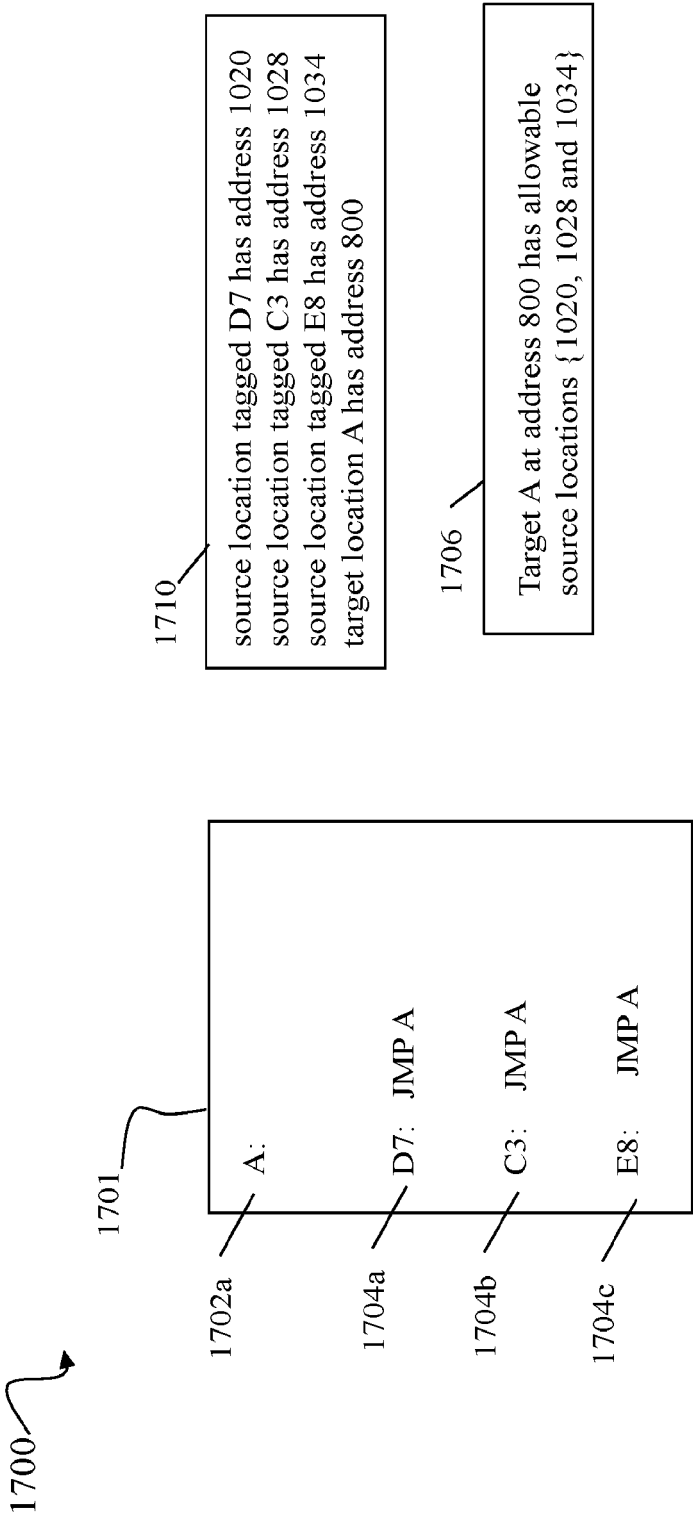


FIG. 83

90/90

1720 

1721	source set = {};
1722	for each source that is allowed to transfer control to target:
1723	t = allocate new CFI source tag;
1724	tag the source with t;
1725	source set = source set + {t};
1726	tag target with source set.

FIG. 84