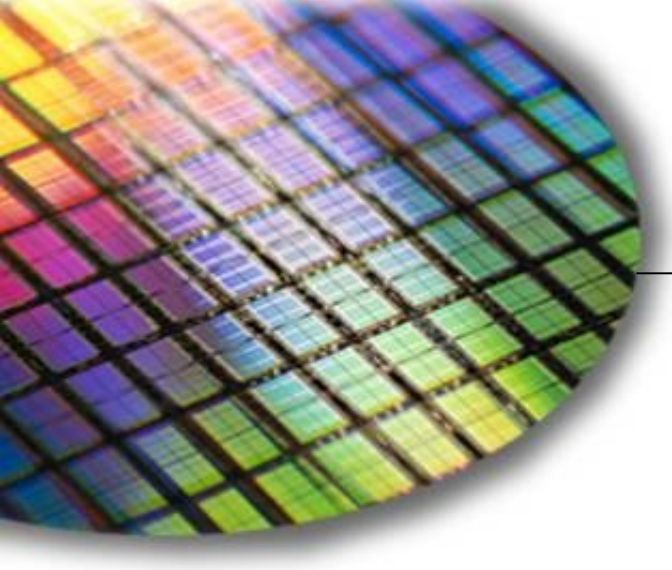
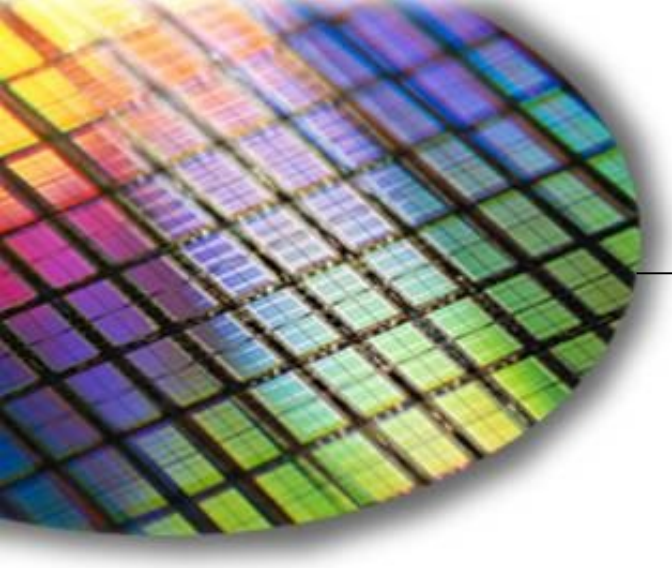


The World Leader in High Performance Signal Processing Solutions

Hardware Design Considerations

Section 19

ANALOG DEVICES



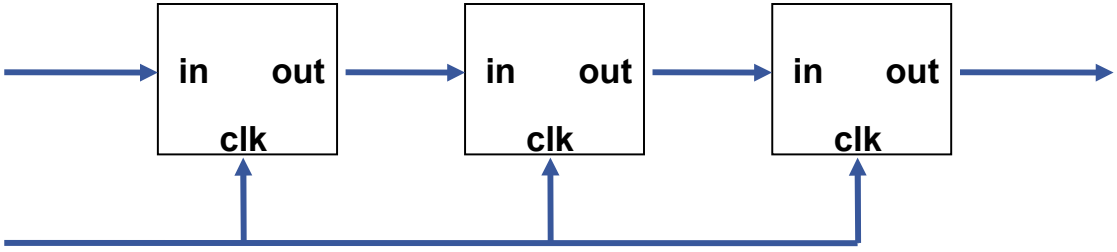
JTAG Port

In-Circuit Emulator Support

- 5-pin JTAG TAP (Test Access Port) allows access to JTAG features
- TAP Controller handles test register event sequencing
- Boundary scan facilitates board-level connectivity testing
  - Up to 32 boundary-scan instructions accommodated

ADSP-BF527 JTAG

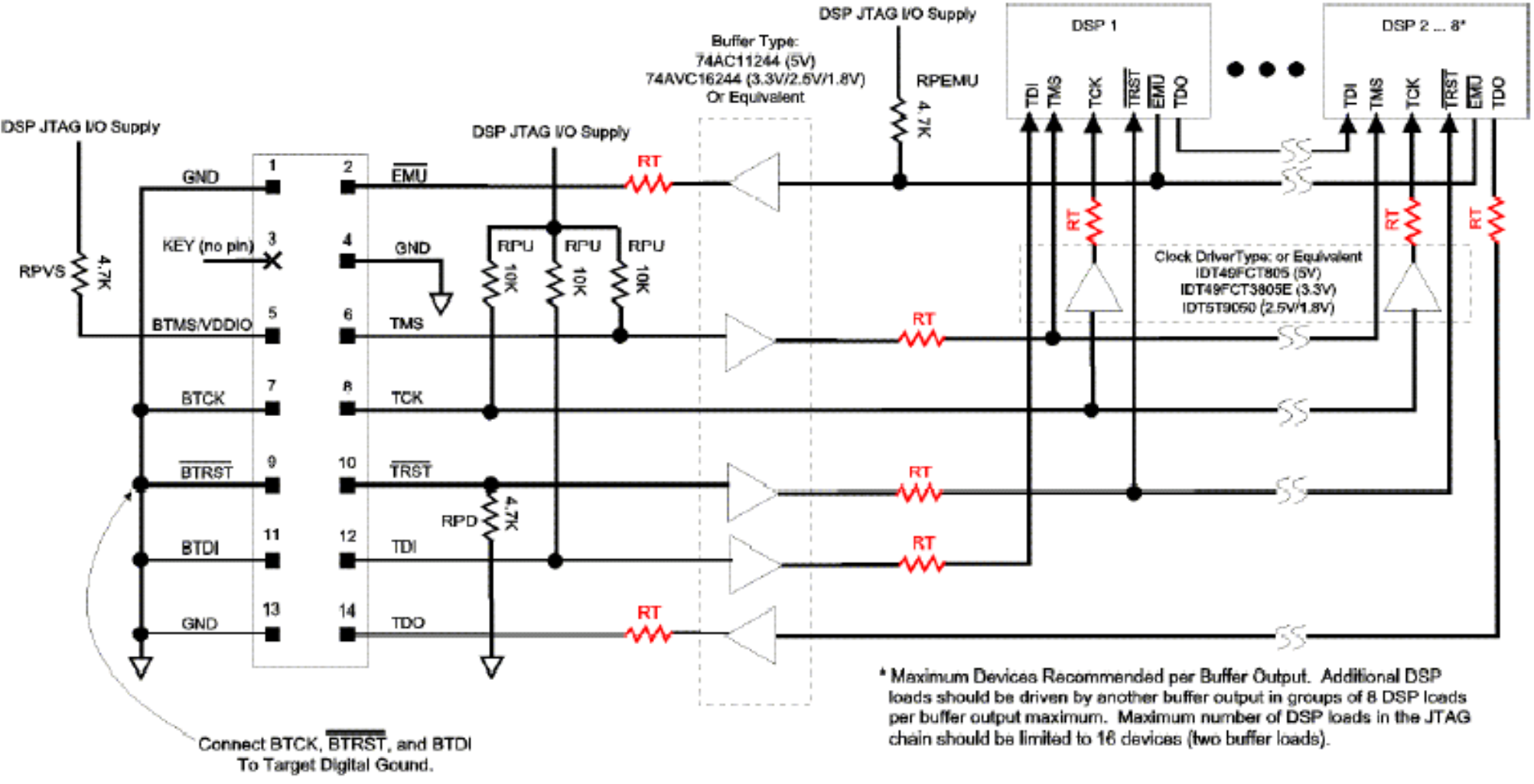
- IEEE standard 1149.1-1990
- Boundary Scan (bsdl files available)



- Read Part Identification Code
  - Analog Devices JEDEC ID is 0x0E5
  - Instruction Register is 5 bits wide
- In Bypass mode behaves like a 1-bit shift register
- Emulation (ADI enhancement)

ADSP-BF527 JTAG Emulation

- Provides communication channel between JTAG emulator hardware and ADI JTAG DSP.



See Application Note EE-68

ANALOG DEVICES

What do you need for ADSP-BF537 Hardware Design?

- ADSP-BF537 DSP Hardware Reference
  - Includes Hardware Examples
- The Datasheet (check the web for latest revision!)
  - Insure that all timing requirements are met!!!!
- The Anomaly Sheet (check the web for latest revision!)
- Use EZ-Kits as Design Example
- Look for Application Notes
- Recommended Reading

High Speed Digital Design - A Handbook of Black Magic
Johnson & Graham
Prentice Hall,Inc.
ISBN 0-13-395724-1

ADSP-BF527 Internal Current Consumption

- **Core Supply IDDINT**
  - **Peak**
  - **Executing from internal memory**
    - TBD
  - **Typical**
  - **Executing from internal memory**
    - 50% are MAC instruction
  - **Power-down All**
  -
- **Peripheral Supply IDDINT**
  - **Typical**
  - **Power-down**

Calculate Power Dissipation

- **Total Power Dissipation**

$$\overline{P_{Total}} = \overline{P_{Intern}} + \overline{P_{Extern}}$$

- **Internal Power Dissipation**

$$\overline{P_{Intern}} = V_{DDINT} \cdot \overline{I_{DDINT}}$$

- **External Power Dissipation**

  Capacitive Bus Load (BF537: CIN max = x pF)
  VDDEXT = 3.3V

$$\overline{P_{Extern}} = V_{DDEXT}^{2} \cdot \sum_{Pins} f \cdot C$$

V_{DDINT} = 1.2V

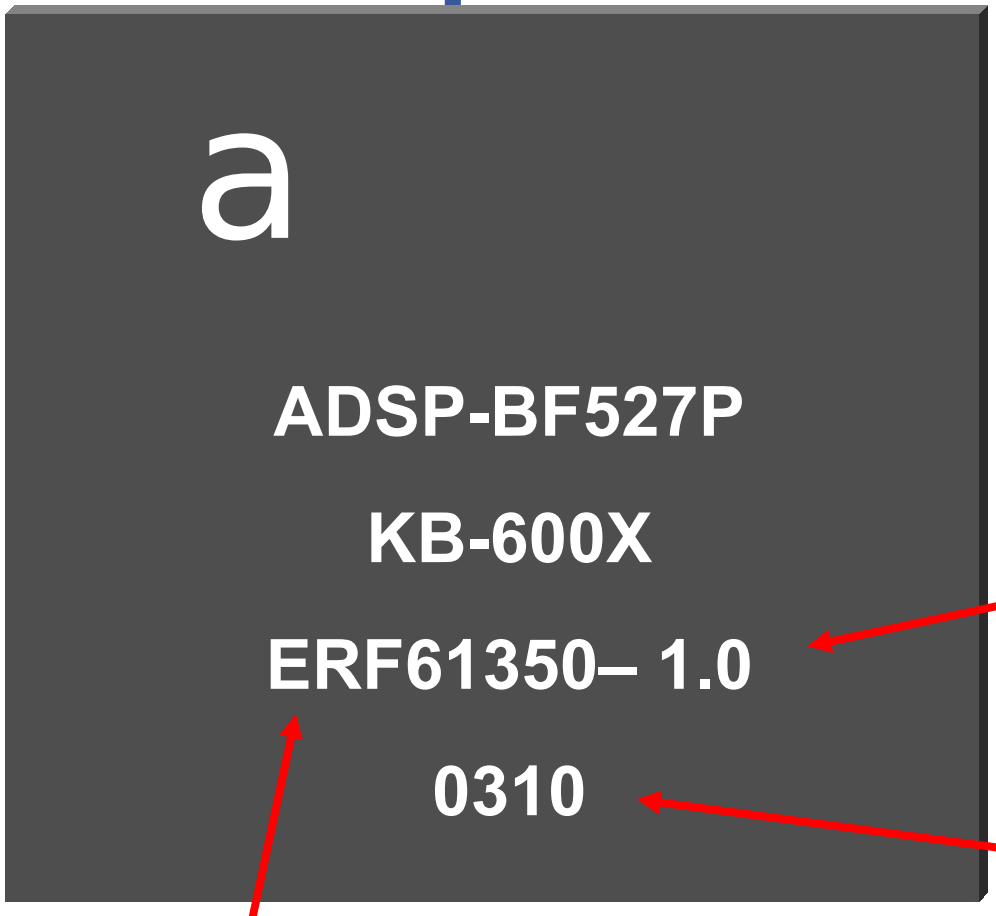
Typical I_{DDINT} is specified in the Datasheet and depends on operating mode

Reference Application Note for Detailed Discussion:
"Estimating Power for ADSP-BF533 Blackfin® Processors (EE-229)"

High-Speed Design Issues

- **Awareness of High Speed Digital Design**
  - Signal Integrity on Wires and Connectors
  - Noise generated by Crosstalk
  - EMI / EMC Compliance
- **Bypassing Capacitors (Ceramic)**
  - 100nF decoupling capacitors between VddInt and Gnd
  - 100nF decoupling capacitors between VddExt and Gnd
  - One 100nF capacitor at the power supply connector of the board
- **Grounding and Shielding**
  - Dedicated VDD and GND supply plane recommended
  - No Ground Loops
  - Signal Return Path as short as possible
  - No Signal Routing over Split Planes

Information printed on the package



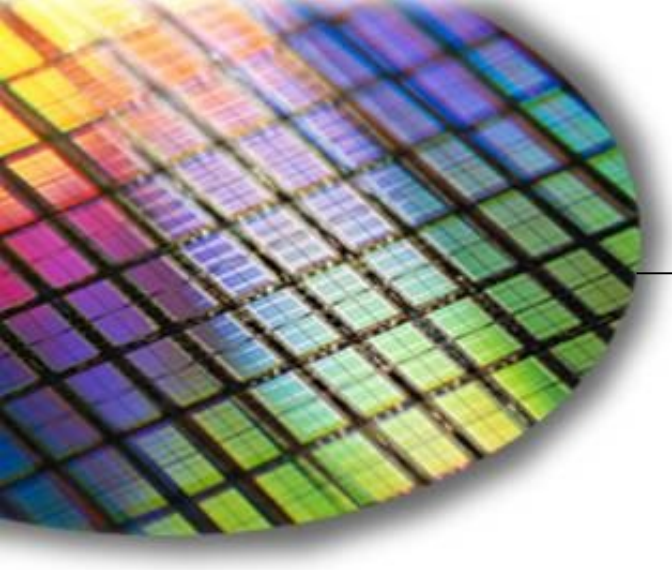
Chip Revision 1.0

Date Code:
10th week of 2003

Lot ID

"E" Wafer Fab Code = TSMC Taiwan

"R" Assembly Location Code = STATS Singapore



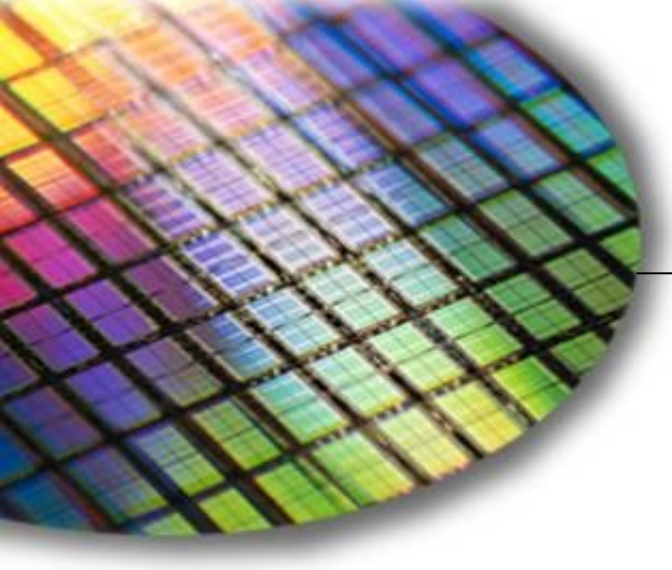
The World Leader in High Performance Signal Processing Solutions

Debug Registers

ANALOG DEVICES

Advanced Support for Embedded Debug

- **Hardware Breakpoints**
  - 6 Instruction and 2 Data Watchpoints
- **Performance Monitor Unit**
  - Counters for cycles and occurrences of specific activities
- **Execution Trace Buffer**
  - Stores last 16 non-incremental PC values



The World Leader in High Performance Signal Processing Solutions

Reference Material

System Design

ANALOG DEVICES

Hardware Breakpoints

- 8 Watchpoints for Instruction/Data Address Comparison
  - 6 Instruction Watchpoints, 2 Data Watchpoints
  - Can be used as 4 Watchpoint ranges instead (3 instruction address ranges + 1 data address range)
- Allow conditional program halting, based on user-specified conditions
  - Memory accesses within a specified range
  - Data loaded into register
  - Stack activity
- Counter allows tracking of watchpoint events
- Ability to combine events
  - Break ONLY when ALL or when ANY enabled events occur

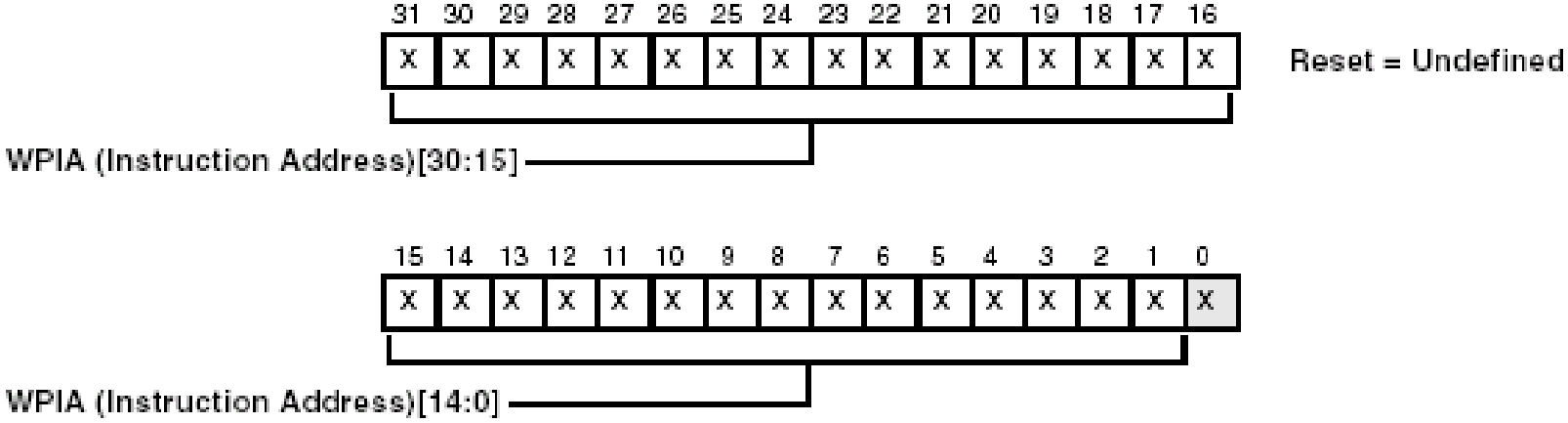
Watchpoint Control Registers

The Watchpoint Unit contains these MMRs, which are accessible in Supervisor and Emulator modes:

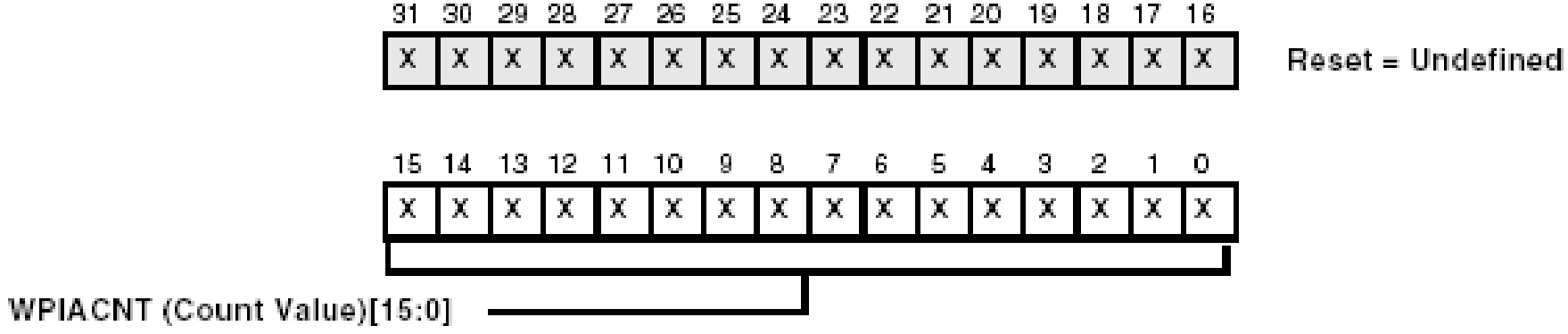
- The Watchpoint Status register (`WPSTAT`)
- Six Watchpoint Instruction Address registers (`WPIA[5:0]`)
- Six Watchpoint Instruction Address Count registers (`WPIACNT[5:0]`)
- The Watchpoint Instruction Address Control register (`WPIACTL`)
- Two Watchpoint Data Address registers (`WPDA[1:0]`)
- Two Watchpoint Data Address Count registers (`WPDACNT[1:0]`)
- The Watchpoint Data Address Control register (`WPDACTL`)

Watchpoint Instruction Address and Count Registers

Instruction Watchpoint Address Registers (WPIAn)



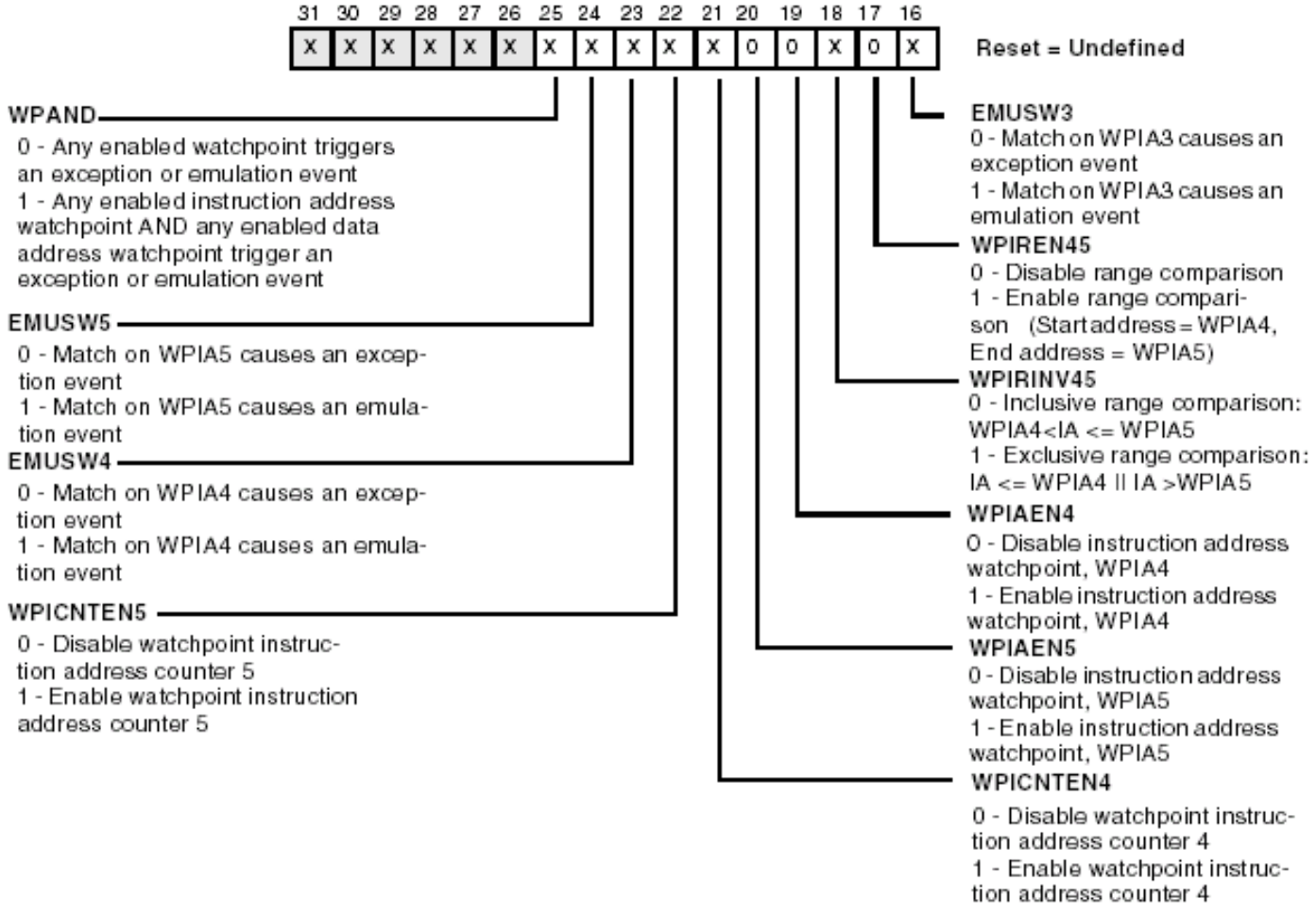
Instruction Watchpoint Address Count Registers (WPIACNTn)



Watchpoint Instruction Address Control Register

Instruction Watchpoint Address Control Register (WPIACTL)

In range comparisons, IA = instruction address.



- **Bits 0-15 of WPIACTL (not shown) control remaining instruction watchpoints**

Watchpoint Data Registers

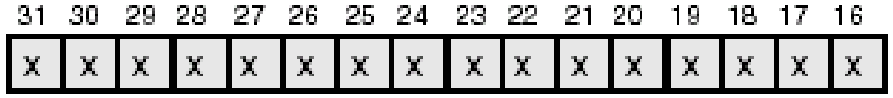
- **Analogous set of registers exist for two Data Watchpoints**

Each data watchpoint is controlled by four bits in the WPDACTL register:

Bit Names	Description
WPDACCx	Determines whether the match should be on a read or write access.
WPDSRCx	Determines which DAG the unit should monitor.
WPDCNTENx	Enables the counter that counts the number of address matches. If the counter is disabled, then every match causes an event.
WPDAENx	Enables the data watchpoint activity.

Watchpoint Status Register

Watchpoint Status Register (WPSTAT)



Reset = Undefined

STATDA1

0 - WPDA1 not matched
1 - WPDA1 matched.

STATDA0

0 - Neither WPDA0 nor the WPDA0 to WPDA1 range matched
1 - WPDA0 matched or the WPDA0 to WPDA1 range matched

STATIA5

0 - WPIA5 not matched
1 - WPIA5 matched.

STATIA4

0 - Neither WPIA4 nor the WPIA4 to WPIA5 range matched
1 - WPIA4 matched or the WPIA4 to WPIA5 range matched

STATIA0

0 - Neither WPIA0 nor the WPIA0 to WPIA1 range matched
1 - WPIA0 matched or the WPIA0 to WPIA1 range matched

STATIA1

0 - WPIA1 not matched
1 - WPIA1 matched

STATIA2

0 - Neither WPIA2 nor the WPIA2 to WPIA3 range matched
1 - WPIA2 matched or the WPIA2 to WPIA3 range matched

STATIA3

0 - WPIA3 not matched
1 - WPIA3 matched

Status bits are sticky and are all cleared upon write of any value to the register

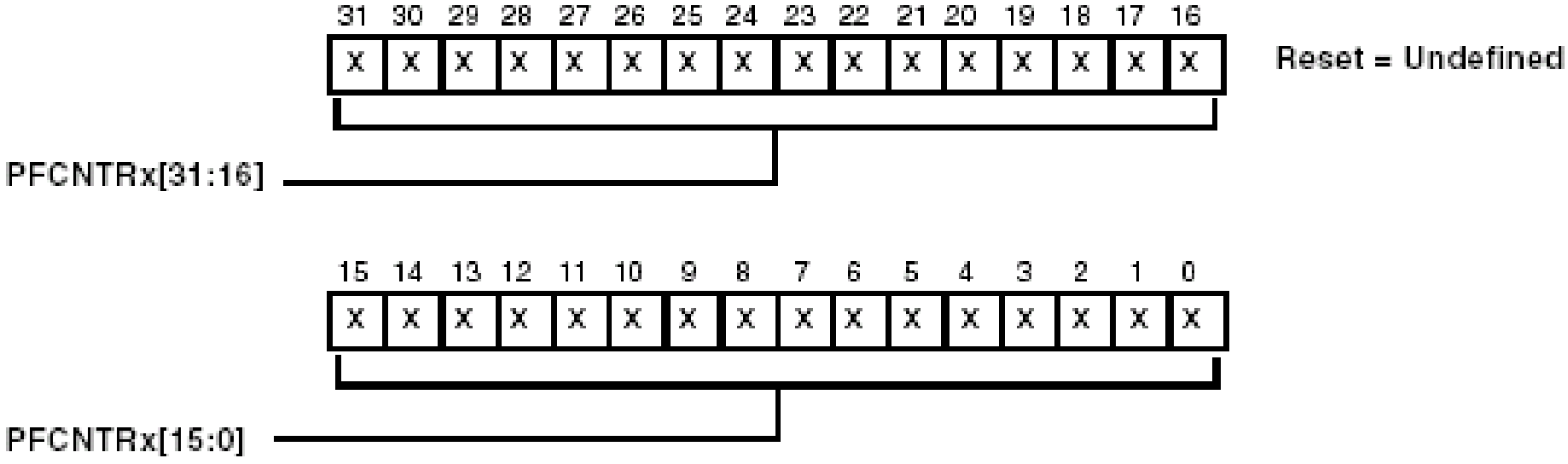
Performance Monitoring

- Dedicated registers available for system tuning and load balancing
- Includes two 32-bit counters, and registers to count number of cycles or occurrences of an event
  - Unit accesses (MAC0, MAC1, DAG0, DAG1)
  - Branches and exceptions
  - Memory conflicts
  - Loads/stores (8/16/32-bit)
  - Cache hits and misses
  - Interrupt latencies

Performance Monitoring Registers

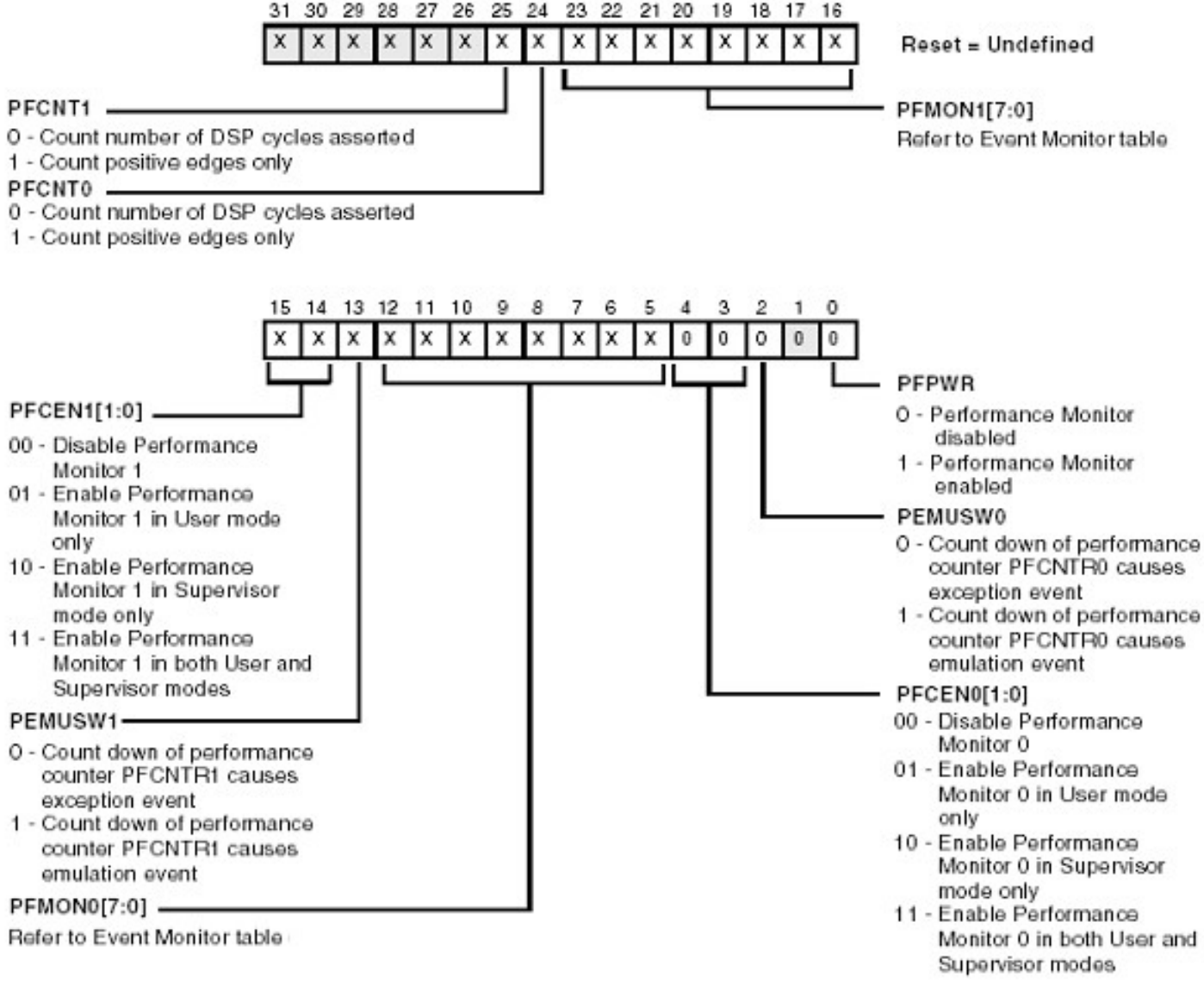
- Two Performance Monitor Counters exist

Performance Monitor Counter Registers (PFCNTRn)



Performance Monitor Control

Performance Monitor Control Register (PFCTL)



What can be counted?

PFMONx Fields	Events That Cause the Count Value to Increment
0x00	Loop 0 iterations
0x01	Loop 1 iterations
0x02	Loop buffer 0 not optimized
0x03	Loop buffer 1 not optimized
0x04	PC invariant branches
0x06	Conditional branches
0x09	Total branches (calls, returns, branches, but not interrupts)
0x0A	Stalls due to CSYNC, SSYNC
0x0B	EXCPT instructions
0x0C	CSYNC, SSYNC instructions
0x0D	Committed instructions
0x0E	Interrupts taken
0x0F	Misaligned address violation exceptions
0x10	Stall cycles due to read after write hazards on DAG registers
0x13	Stall cycles due to RAW data hazards in computes
0x80	Processor stalls to memory
0x81	Data memory stalls to processor hidden by processor stall
0x82	Data memory store buffer full stalls

PFMONx Fields	Events That Cause the Count Value to Increment
0x83	Data memory write buffer full stalls due to high to low priority code transition
0x84	Data memory store buffer forward stalls due to lack of committed data from processor
0x85	Data memory fill buffer stalls
0x86	Data memory array or TAG collision stalls (DAG to DAG, or DMA to DAG)
0x87	Data memory array collision stalls (DAG to DAG or DMA to DAG)
0x88	Data memory stalls
0x89	Data memory stalls sent to processor
0x8A	Data memory cache fills completed to Bank A
0x8B	Data memory cache fills completed to Bank B
0x8C	Data memory cache victims delivered from Bank A
0x8D	Data memory cache victims delivered from Bank B
0x8E	Data memory cache high priority fills requested
0x8F	Data memory cache low priority fills requested
0x90	Code memory fetches postponed due to DMA collisions (minimum count of two per event)
0x91	Code memory TAG stalls (cache misses, or FlushI operations, count of 3 per FlushI). Note code memory stall results in a processor stall only if instruction assembly unit FIFO empties.
0x92	Code memory fill stalls (cacheable or non-cacheable). Note code memory stall results in a processor stall only if instruction assembly unit FIFO empties.
0x93	Code memory 64 bit words delivered to processor instruction assembly unit

Performance Monitoring: Cycle Counters

- All cycles are counted
- Counters stop during Emulation
- Can be read in all operating modes
- Can only be written in the Emulator or Supervisor modes
- Must be enabled to use

Example Code for Turning on Cycle Counters

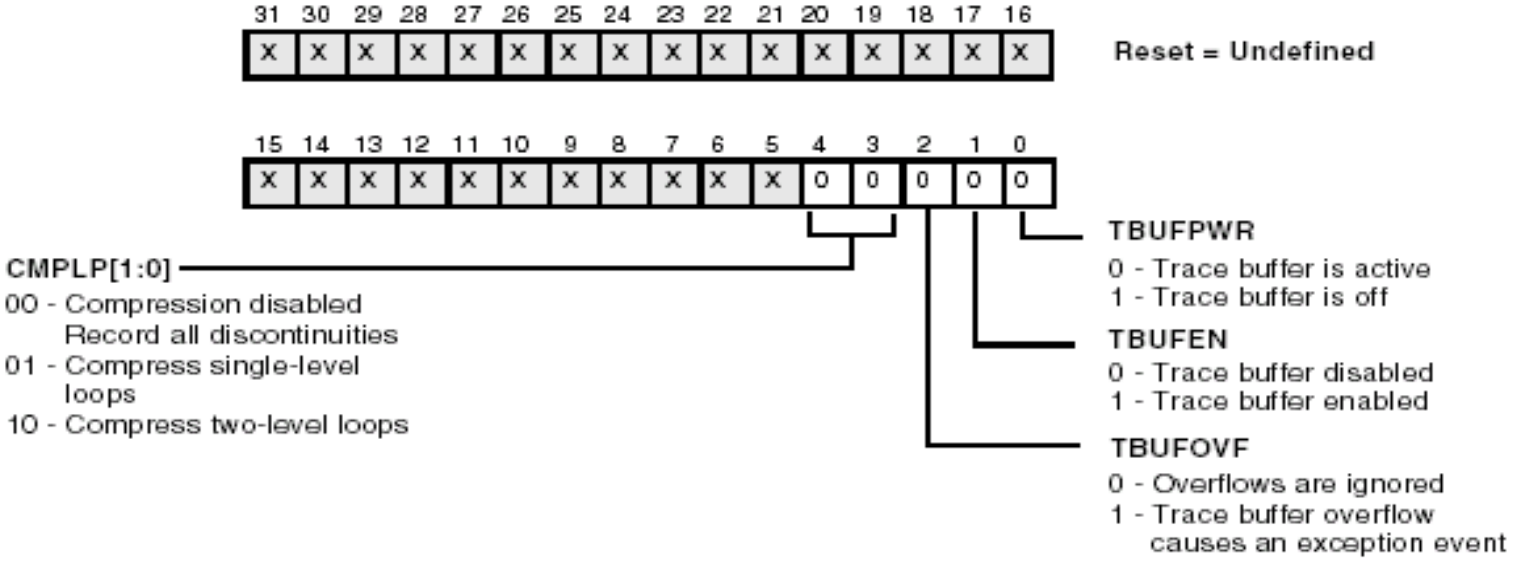
```
R2 = 0;
CYCLES = R2;
CYCLES2 = R2;
R2 = SYSCFG;
BITSET(R2,1);
SYSCFG = R2;
/* Insert code to be benchmarked here. */
```

Execution Trace

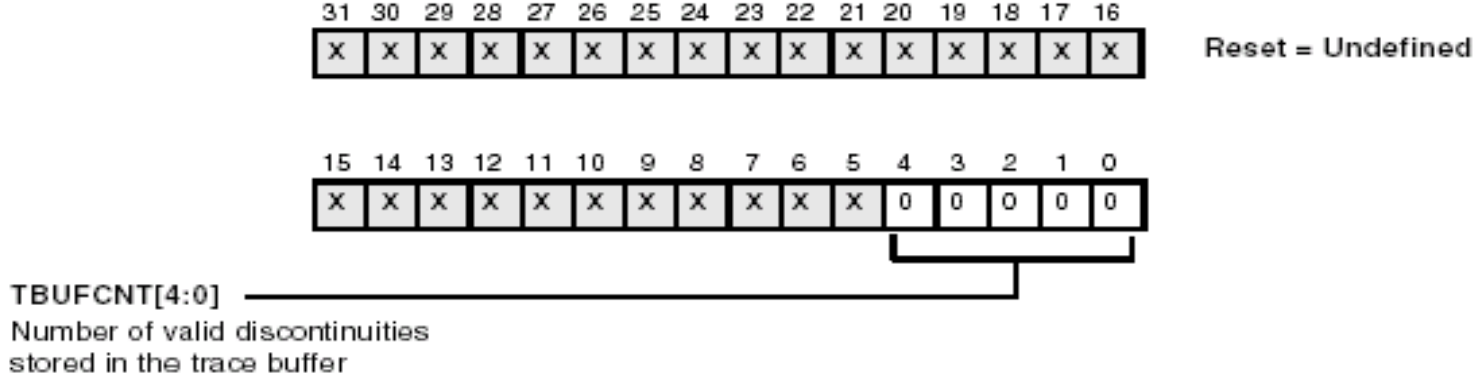
- 16-entry trace buffer stores changes in program flow (non-contiguous addresses)
  - Jumps
  - Calls
  - Interrupts
- Used for recent-history examination
- Zero-overhead hardware loops register only once in the trace buffer

Trace Buffer Registers

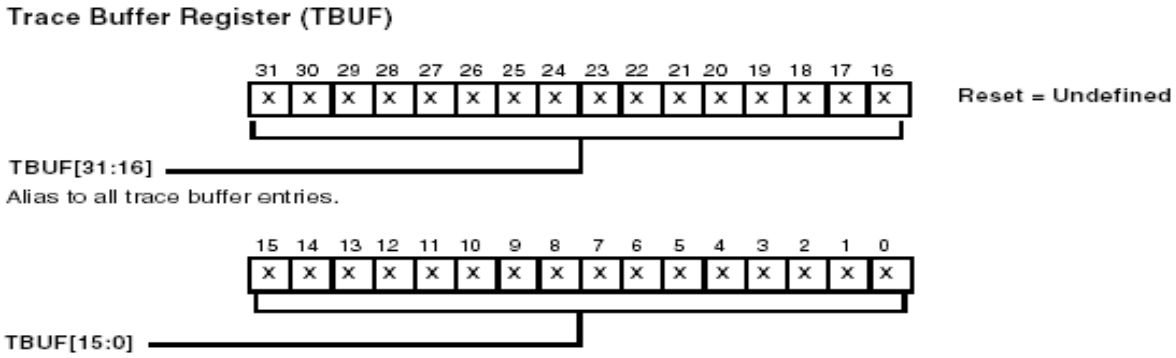
Trace Buffer Control Register (TBUFCTL)



Trace Buffer Status Register (TBUFSTAT)



Accessing the Trace Buffer



- The first read returns the latest branch target address.
- The second read returns the latest branch source address.

Example Code for Recreating Execution Trace in Memory

```
[--sp] = (r7:7, p5:2); /* save registers used in this routine */
p5 = 32;  /* 32 reads are needed to empty TBUF */
   p2.l = buf;  /* pointer to the header (first location) of the
   software trace buffer */
   p2.h = buf;  /* the header stores the first available empty buf
   location for subsequent trace dumps */

p4 = [p2++];  /* get the first available empty buf location from
the buf header */
p3.l = TBUF & 0xffff;  /* low 16-bits of TBUF */
p3.h = TBUF >> 16;  /* high 16-bits of TBUF */

lsetup(loop1_start, loop1_end) lc0 = p5;
loop1_start: r7 = [p3];  /* read from TBUF */
loop1_end: [p4++] = r7;  /* write to memory and increment  */
[p2] = p4;  /* pointer to the next available buf location is
saved in the header of buf */
(r7:7, p5:3) = [sp++];  /* restore saved registers */
```