

Knowledgebase > Tracing > How can I measure clock cycles per instruction (CPI) using TRACE32?

How can I measure clock cycles per instruction (CPI) using TRACE32?

2025-08-21 - Comments (0) - Tracing

There are mainly two methods to obtain **CPI** (clock cycles per instruction) or **MIPS** (Millions of Instructions Per Second) in TRACE32.

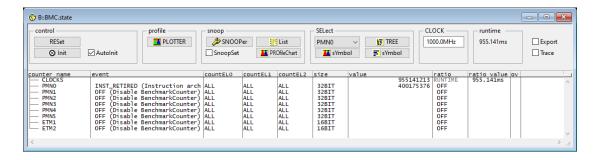
BenchMark Counters

The first method is **BMC (BenchMark Counters)**, which in the case of Arm processors correspond to the PMU (Performance Monitoring Unit). BMC is supported by the TRACE32 BMC.* command group. The results are directly obtained from the core counters, which makes BMC the recommended primary method for measuring CPI or MIPS.

In addition, BMC often provides counters for cache hits and misses, which may also be useful in the context of CPI analysis. To check whether BMC is supported for your processor architecture, refer to the *Processor Architecture Manual* for your target in main.pdf. Generic BMC commands are documented in the General Commands Reference Guide B.

Example for Arm Cortex-A72

BMC.state
BMC.AutoInit ON
BMC.CLOCK 1000MHz
BMC.PMN0.EVENT INST_RETIRED
BMC.PMN0.RATIO X/TIME
BMC.Init
Go
WAIT 1.s
Break



Using Trace-Based Measurments

The second method is to use **trace-based measurements** to obtain **CPI** or **MIPS** values. This approach is mainly useful for metrics not covered by BMC counters. CPI can be statistically analyzed from trace data using commands such as **Trace.STATistic.CYcle**, which provides average clocks per instruction calculated from traced instructions and cycles.

Trace.STATistic.CYcle

MIPS	KWINST K	ALL	4096.				
	records:	1628.	instr:	6893.	T		
	time:	10.486us	instr/second: 657.356578MHz				
	clocks:	10486.	cpi:	1.52			
	cycles	bytes	cycles/second	bytes/second			
flow fetch		27572.	657.356578MHz	2629.426315MB	П		
flow read		14490.	177.857249MHz	1381.850692MB			
flow write	250.	1809.	23.841454MHz	172.516763MB			
	cycles	bytes	cycles/second	bytes/second			
bus read		0.	0.Hz	0.B			
bus write		0.	0.Hz	0.B			
	instructions	ratio	frequency		Ш		
instr		100.000%	657.356578MHz				
cond instr pass		0.000%	0.Hz				
cond instr fail		0.000%	0.Hz				
load instr		26.084%	171.467739MHz				
store instr		2.814%	18.500968MHz				
load/store instr		0.000%	0.Hz				
uncond branch		2.582%	16.975115MHz				
cond branch		22.036%	144.860676MHz				
	branches	ratio	frequency		4		
uncond dir		1.581%	10.394874MHz				
uncond indir		1.001%	6.580241MHz				
cond not taken		0.812%	5.340485MHz				
cond dir taken		21.224%	139.52019MHz				
ond indir taken calls		0.000%	0. Hz 6. 866338MHz				
calls returns		1.044%	5.721949MHz				
		0.870%					
traps		0.000% 0.014%	0.Hz 95.365KHz				
interrupts				-1			
idles	number 1.	0.000%	frequency 95.365KHz	clocks	-		
fifofulls		0.000%	93.303KHZ				
trace gaps		0.000%					

The MIPS.* commands can additionally be used to analyze the MIPS. The system can be analyzed under different aspects: workload per task, workload per high-level language line, workload per specified functional group etc. The following command displays for instance a numerical analysis per function.

MIPS.STATistic.Func

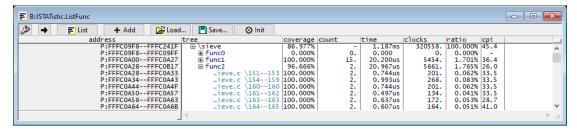
9 <mark>iii</mark> # →	🚂 Detai	led 📳 List	t 😾 Fun	cs 🙀 Nest	ing 🖺 F	lat 🔃 Tree		409	96.			
		funcs: 36.	1	otal:	7067.							
		total	min	max	avr	count	intern%	196	2%	5%	10%	20%
aarch64_v8\sieve			547.				15.480%					
\sieve\test_cond			14.				1.075%	-				
aarch64_v8\sieve			139.				3.084%					
arch64_v8\sieve\t	Func2a	194.	97.				2.745%					
arch64_v8\sieve\			76.				2.150%					
arch64_v8\sieve\t			88.				2.490%					
arch64_v8\sieve\t			109.				3.084%					
sieve\ init_linke			445				12.593%					
aarch64_v8\sieve		38.	19.	19.	. 19	. 2.	0.537%					
aarch64_v8\sieve			2.				0.056%					
aarch64_v8\sieve	func5	22.	11.	11.	. 11		0.311%	+				
aarch64_v8\sieve	func6	48.	24.	24.			0.679%	+				
aarch64_v8\sieve	func7	48.	24.	24.	. 24	. 2.	0.679%	+				
aarch64_v8\sieve	Func8	382.	191.	191.	. 191	. 2.	5.405%			_		

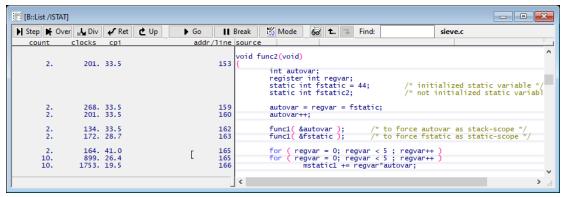
Refer for more information about the ${\tt MIPS.*}$ command group to the ${\tt General\ Commands\ Reference\ Guide\ M}.$

Moreover, you can use the ${\tt ISTATistic.*}$ command group to obtain detailed information for individual instructions when a program trace is available. However, it should be noted that for "high-end" cores this method may not provide cycle-accurate results for every single instruction.

Example:

ISTATistic.ADD ISTATistic.ListFunc List /ISTAT





Refer for more information about the ${\tt ISTATistic.*}$ command group to the ${\tt General\ Commands\ Reference\ Guide\ I.}$

Note

The trace results must not include errors, such as FIFOFULLs or FLOWERRORs, in order to provide reliable results. Refer for more information to <u>HARDERRORs</u>, <u>FIFOFULLs</u> and <u>FLOWERRORs</u> in the trace.