MIPS

- The MIPS rating is a poor metric that fails for the same reason as the clock rate, i.e., *it doesn't account for the efficiency of the instructions*
- MIPS tells you only how fast a computer executes instructions, but *doesn't tell you what is actually achieved by the instructions being executed*.
- Consider the following example of computation on two computers *A* and *B*, where computer *A* has a load/store architecture without a multiplier and computer *B* has a memory-to-register architecture with a multiplier.
  - Both computers evaluate the expression $z = 4 \times (x + y)$.

Computer A (LOAD/STORE) Without a multiplier

```
LDR   r1,[r0]   ;load x
LDR   r2,[r0,4] ;load y
ADD   r2,r1,r2  ;x+y
ADD   r2,r2,r2  ;2(x+y)
ADD   r2,r2,r2  ;4(x+y)
STR   r2,[8,r0] ;store z
```

Computer B (memory-register) With a multiplier

```
LDR   r1,[r0]    ;load x
ADD   r1,[r0,4]  ;x+y
MUL   r1,#4      ;4(x+y)
STR   r1,[8,r0]  ;store z
```

MIPS

- Suppose ***computers A and B have the same MIPS***, i.e., executing instructions at the same speed, on average.
- If you relied solely on MIPS as a metric of performance (i.e., without knowing what is actually achieved by the instructions being executed), you'd conclude that the two computers offer the same performance.
- As you can see, ***computer B is faster than computer A*** because only four instructions are required to do the work (*this argument is based on the assumption that all instructions take the same time*).

Computer A (LOAD/STORE)
Without a multiplier

```
LDR   r1,[r0]   ;load x
LDR   r2,[r0,4] ;load y
ADD   r2,r1,r2  ;x+y
ADD   r2,r2,r2  ;2(x+y)
ADD   r2,r2,r2  ;4(x+y)
STR   r2,[8,r0] ;store z
```

Computer B (memory-register)
With a multiplier

```
LDR   r1,[r0]    ;load x
ADD   r1,[r0,4]  ;x+y
MUL   r1,#4      ;4(x+y)
STR   r1,[8,r0]  ;store z
```

MIPS

- Suppose ***computers A and B completing the below tasks exactly at the same time***:
- This suggests that both machines perform the same (*this argument is based on the assumption that both machines successfully perform the same task at the same time*).
- However, from the MIPS view point, ***computer A is 50% better than Computer B***, as it can execute 50% more instructions than computer B at the same given time.

Computer A (LOAD/STORE)
Without a multiplier

```
LDR   r1,[r0]   ;load x
LDR   r2,[r0,4] ;load y
ADD   r2,r1,r2  ;x+y
ADD   r2,r2,r2  ;2(x+y)
ADD   r2,r2,r2  ;4(x+y)
STR   r2,[8,r0] ;store z
```

Computer B (memory-register)
With a multiplier

```
LDR   r1,[r0]    ;load x
ADD   r1,[r0,4]  ;x+y
MUL   r1,#4      ;4(x+y)
STR   r1,[8,r0]  ;store z
```

MIPS

- In practice, computer *A* might be faster than *B* (i.e., A might has a better MIPS than computer B) because memory-to-register architectures are slower than register-to-register architectures.

Computer A (LOAD/STORE) Without a multiplier

```
LDR   r1,[r0]   ;load x
LDR   r2,[r0,4] ;load y
ADD   r2,r1,r2  ;x+y
ADD   r2,r2,r2  ;2(x+y)
ADD   r2,r2,r2  ;4(x+y)
STR   r2,[8,r0] ;store z
```

Computer B (memory-register) With a multiplier

```
LDR   r1,[r0]    ;load x
ADD   r1,[r0,4]  ;x+y
MUL   r1,#4      ;4(x+y)
STR   r1,[8,r0]  ;store z
```

MIPS

Pipelining is not considered here

- The duration of a single instruction is

$$t_{cycle} \times c$$

where c is the number of machine cycles required to execute the instruction and t_{cycle} is the cycle time (usually the clock period).

- The total execution time for a program is given by:

$$t_{execution} = t_{cycle} \times \sum n_i \, c_i$$

where n_i is the number of times instruction i occurs in the program and c_i is the number of cycles required to execute instruction i.

- If we plug this formula into the equation for MIPS, we get

$$MIPS = \frac{n/10^6}{t_{execute}}$$

$$MIPS = \frac{n/10^6}{t_{cycle} \times \sum n_i c_i} = \frac{n}{10^6 \times t_{cycle} \times \sum n_i c_i}$$

CPI

- The *average number of* ***clock cycles per instruction*** is determined by the instruction mix (i.e., the relative number of 1-cycle, 2-cycle, 3-cycle instructions etc.)

$$CPI = \sum_{i=1}^{N} f_i \times c_i$$

  - f_i is the fraction of instructions taking c_i clock cycles to execute.

$$\sum f_i = 1$$

  - the value of i ranges from 1 (one instruction per cycle) to N.
  - The value of N is the length of the longest instruction in terms of clock cycles.

© 2014 Cengage Learning Engineering. All Rights Reserved.

CPI

- A benchmark runs on a hypothetical processor to give the results in Table 6.3.
- We can obtain the *average number of clock cycles per instruction* as shown in Table 6.4.

The sum of all relative frequencies = 100%

TABLE 6.3 Relative Instruction Frequency and Cycle Count for a Computer

Machine Operation	Relative Frequency	Cycles per Instruction
Arithmetic/logical instruction	53%	1
Register load operation	20%	4
Register store operation	7%	2
Unconditional branch instruction	12%	1
Conditional branch instruction	8%	2

TABLE 6. 4 Calculating the Average CPI for the System in Table 6.3

Machine Operation	Frequency	Cycles	CPI
Arithmetic/logical instruction	53%	1	0.53
Register load operation	20%	4	0.80
Register store operation	7%	2	0.14
Unconditional branch instruction	12%	1	0.12
Conditional branch instruction	8%	2	0.16
Average cycles per instruction			1.75

This is between 1 and 4

CPI

- We can also relate the relative instruction frequencies to the ***percentage of time an instruction class takes*** by multiplying the instruction class frequency by the number of cycles and dividing the result by the average cycles per instruction (i.e., 1.75), Table 6.5.

- In Table 6.5 the register load instruction takes up 20% of the code, but is responsible for 45.71% of the processor cycle time; this is telling us that loading registers from memory is expensive.

TABLE 6.5 Calculating the Average Time Spent Executing Each Instruction Class

Machine Operation	Frequency	Average Cycles	Average Time
Arithmetic/logical instruction	53%	$1 \times .53 = 0.53$	30.29%
Register load operation	20%	$4 \times .2 = 0.80$	45.71%
Register store operation	7%	$2 \times .07 = 0.14$	8.00%
Unconditional branch instruction	12%	$1 \times 0.12 = 0.12$	6.86%
Conditional branch instruction	8%	$2 \times .08 = 0.16$	9.14%

© 2014 Cengage Learning Engineering. All Rights Reserved.

CPI and MIPS

- A program is running on a computer with the following parameters

Clock cycle time = 20 ns (1 ns = 10^{-9} second)

Instructions with 1 clock cycle	15%
Instructions with 2 clock cycles	40%
Instructions with 3 clock cycles	30%
Instructions with 4 clock cycles	10%
Instructions with 5 clock cycles	5%

What is the MIPS rating of this computer?

Clock cycle time = 20 ns
Clock rate = 1/ Clock cycle time
= 1 / 20 ns
= 10^9 / 20
= 10^6 × 1000/20
= 50 × 10^6
= 50 Mega Hertz

Answer

The average Clocks Per Instruction (CPI) =

$$= 0.15 \times 1 + 0.40 \times 2 + 0.30 \times 3 + 0.10 \times 4 + 0.05 \times 5$$

$$= 0.15 + 0.80 + 0.90 + 0.40 + 0.25 = 2.50 \text{ CPI}$$

If the clock period is 20ns, the average instruction takes 2.5 × 20 ns = 50 ns

The number of instructions per second is 1/50ns = 10^9 /50

= 10^6 × 1000 /50

= 20 MIPS.

CPI and MIPS

In the previous example, if the clock time is reduced by 5%, it would require one extra cycle to execute any instruction in the first group (i.e., the 15% group). Would that be a good idea?

Answer

Clock cycle time = 20 ns × **0.95 = 19 ns**

Faster clock

Instructions with 1 **+ 1** clock cycle	15%
Instructions with 2 clock cycles	40%
Instructions with 3 clock cycles	30%
Instructions with 4 clock cycles	10%
Instructions with 5 clock cycles	5%

Clock cycle time = 19 ns
Clock rate = 1/ Clock cycle time
= 1 / 19 ns
= 10^9 / 19
= 10^6 × 1000/19
= 52.63 × 10^6
= 52.63 Mega Hertz

The average Clocks Per Instruction (CPI) =

= 0.15 × **2** + 0.40 × 2 + 0.30 × 3 + 0.10 × 4 + 0.05 × 5

= **0.30** + 0.80 + 0.90 + 0.40 + 0.25 = **2.65 CPI**

Higher CPI

More time

If the clock period is **19ns**, the average instruction takes **2.65** × **19** ns = **50.35**ns

The number of instructions per second is 1/ **50.35** ns = 10^9 /**50.35**

= 10^6 × 1000 / **50.35**

= **19.86** MIPS.

Lower MIPS

It is NOT worth it

CPI and MIPS

In the previous example, if the clock time is reduced by 10%, it would require one extra cycle to execute any instruction in the first group (i.e., the 15% group). Would that be a good idea?

Answer

Faster clock

Clock cycle time = 20 ns × **0.90 = 18 ns**

Instructions with 1 **+ 1** clock cycle	15%
Instructions with 2 clock cycles	40%
Instructions with 3 clock cycles	30%
Instructions with 4 clock cycles	10%
Instructions with 5 clock cycles	5%

Clock cycle time = 18 ns
Clock rate = 1/ Clock cycle time
= 1 / 18 ns
= 10^9 / 18
= 10^6 × 1000/18
= 55.56 × 10^6
= 55.56 Mega Hertz

Higher CPI

Less time

The average Clocks Per Instruction (CPI) =

= 0.15 × **2** + 0.40 × 2 + 0.30 × 3 + 0.10 × 4 + 0.05 × 5

= **0.30** + 0.80 + 0.90 + 0.40 + 0.25 = **2.65 CPI**

If the clock period is **18ns**, the average instruction takes **2.65** × **18** ns = **47.7** ns

The number of instructions per second is 1/ **47.7** ns = 10^9 /**47.7**

= 10^6 × 1000 / **47.7**

= **20.96** MIPS.

It is worth it

Higher MIPS

CPI and MIPS

- MIPS is sensitive to the way in which a compiler generates code.
- Consider the following example.
  - A program is compiled to run on a computer and the compiler generates *2.0 million* one-cycle instructions and *1.0 million* two-cycle instructions.
  - A different compiler generates code for this problem but with *1.5 million* one-cycle instructions and *1.2 million* two-cycle instructions.

If we assume that the cycle time is 10 ns, the time taken is given by:

- The time required to execute the first code is

$$\begin{aligned} t_{execution} &= 2 \times 10^6 \times 1 \times 10 \text{ ns} + 1 \times 10^6 \times 2 \times 10 \text{ ns} \\ &= 4 \times 10^6 \times 10 \text{ ns} \\ &= 40 \text{ ms}. \end{aligned}$$

Clock cycle time = 10 ns
Clock rate = 1/ Clock cycle time
= 1 / 10 ns
= 10^9 / 10
= 10^6 × 1000/10
= 100 × 10^6
= 100 Mega Hertz

- Now the time required to execute the second code is:

$$\begin{aligned} t_{execution} &= 1.5 \times 10^6 \times 1 \times 10 \text{ ns} + 1.2 \times 10^6 \times 2 \times 10 \text{ ns} \\ &= 3.9 \times 10^6 \times 10 \text{ ns} \\ &= 39 \text{ ms}. \end{aligned}$$

The second compiler generated faster code

CPI and MIPS

- Now let's evaluate the MIPS for each case.
  - In the first case, the MIPS is given by

    $MIPS = (3 \times 10^6)/(10^6 \times 40 \times 10^{-3}) = (1000 \times 3) / 40$

    $= 75$ MIPS
  - In the second case, the MIPS is given by
  - $MIPS = (2.7 \times 10^6)/(10^6 \times 39 \times 10^{-3}) = (1000 \times 2.7) / 39$

    $= 69.23$ MIPS

The faster execution has a lower MIPS!!!

- This failure of the MIPS metric is unavoidable because instruction ***throughput*** takes no account of how much work each instruction actually performs.

© 2014 Cengage Learning Engineering. All Rights Reserved.

MIPS

Year	Chip	Millions of Instructions per Second
1985	Intel 386DX	11 MIPS at 33 MHz
1992	Intel 486DX	54 MIPS at 66 MHz
1996	Intel Pentium Pro	541 MIPS at 200 MHz
1999	Intel Pentium III	2,054 MIPS at 600 MHz
2003	Intel Pentium 4	9,726 MIPS at 3.2 GHz
2006	Intel Core 2 X6800 (2 core)	27,079 MIPS at 2.93 GHz
2006	Intel Core 2 QX6700 (4 core)	49,161 MIPS at 2.66 GHz
2008	Intel Core i7 920 (4 core)	82,300 MIPS at 2.66 GHz
2011	Intel Core i7 3960X (6 core)	177,730 MIPS at 3.33 GHz
2013	Intel Core i7 4770K (4 core)	133,740 MIPS at 3.9 GHz
2014	Intel Core i7 5960X (8 core)	238,310 MIPS at 3.0 GHz
2015	Intel Core i7 6700K (4 core)	~161,173 MIPS at 4.0 GHz
	iPhone 4S	~5,000 MIPS
	iPhone 5S	~20,500 MIPS
	iPhone 6	~25,000 MIPS

To put things into perspective: *4 seconds of processing by Intel's (2014) i7 5960X would have taken the best 1985 personal computer over 24 hours!!*

MFLOPS

- MFLOPS indicates *millions of floating-point instructions per second*.
- In principal, the same objections to MIPS apply to the MFLOPS metric and therefore you might expect MFLOPS to be as poor an indicator of performance as MIPS.
- MFLOPS is actually a better metric than MIPS because MFLOPS measures the *work done* rather than *instruction throughput*.
  - MIPS counts *all instructions* executed by a computer, many of which perform no useful work in solving a problem (e.g., data movement operations).
  - MFLOPS considers only *floating-point operations* which are at the heart of the algorithm being implemented.

© 2014 Cengage Learning Engineering. All Rights Reserved.

Benchmarks

- The ideal program with which to evaluate a computer is the one you are going to run on that computer.
  - Unfortunately, it's normally impractical to carry out such a test.
  - Moreover, computers execute a mix of programs that changes from moment to moment.
- One approach to benchmarking is based on ***kernels*** or ***fragments of real programs*** that require intense computation, e.g., the LINPACK benchmark
  - LINPACK is a FORTRAN subroutines package that solves mathematical equations, including
    - Linear equations
    - Least Square problems
  - LINPACK has been used in 1970's and early 1980's
  - Currently, it is replaced by LAPACK
- Another approach is to run ***synthetic*** benchmark programs which are constructed to be similar to the type of code that users might actually execute.

Benchmarks

- Benchmarks can be divided into two categories:
  - *fine-grained* and
  - *coarse-grained*.
- The granularity of a benchmark is a function of the object being measured.
  - a benchmark that measures the performance of a complete computer system can be considered coarse-grained,
  - whereas a benchmark that measures the performance of specific instructions, say, branch instructions, can be considered fine-grained.
    - Fine-grained benchmarks a.k.a. micro-benchmark

Benchmarks

- Benchmarks must be interpreted carefully.
- Need to know
  - What is being measured? and
  - How the measurements have been carried out?

- A benchmark that relies very heavily on pure CPU processing power, should not be used to assess the performance of an application involves a large database and much disk-based I/O.

- For many users who surf the internet, the bottleneck is mostly the Internet connection speed.

Benchmarks

- Benchmarks attempt to tell how good a system is at a given workload.

- Yet, they don't tell anything about its incremental performance,
- What happens as the workload is increased? (scalability)
  - Some systems demonstrate a severe drop in performance when the load grows.

SPEC

- Various organizations exist to offer the consumer *unbiased* advice.
- Such an organization has arisen to create benchmarks.

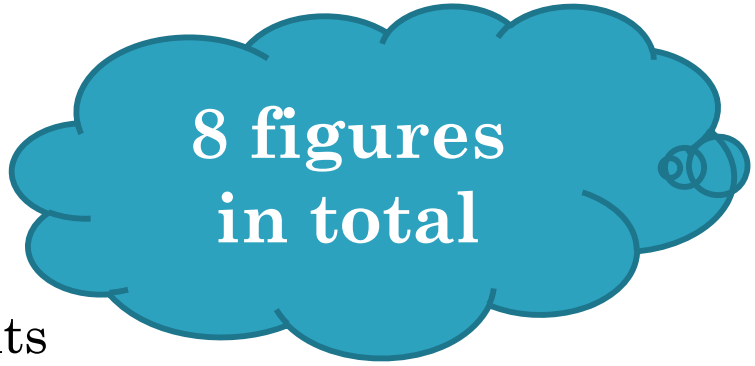
- SPEC, the *Standard Performance Evaluation Corporation*, is a charitable non-profitable organization that's been formed to *establish, maintain and endorse a standardized set of relevant benchmarks that can be applied to the newest generation of high-performance computers*.

- The user is responsible for compiling the benchmarks before they are run.
- SPEC allows two types of compilation.
  - One type of compilation is restrictive in terms of compiler flags/switches and that provides a *base* result.
  - The other type of compilation is more liberal and allows switches to be optimized for each individual program and that provides a *peak* metric.

© 2014 Cengage Learning Engineering. All Rights Reserved.

SPEC

- The figures that are quoted by SPEC include
  - The ***Base*** and the ***Peak*** of
    - Integer operation measurements
    - Floating-point operation measurements
    - Time measurements
    - Rate measurements



© 2014 Cengage Learning Engineering. All Rights Reserved.

SPEC

- SPEC is widely available and it is relatively easy to obtain.
- SPEC provides ***real results on real data*** rather than simply relying on metrics such as clock rate and MIPS or MFLOPS, which do not tell the whole story.

© 2014 Cengage Learning Engineering. All Rights Reserved.

SPEC

- SPEC benchmarks appeared in 1988 with SPEC89, a suite of 10 programs.
- 1992: the next major change (SPEC CPU92)
  - SPEC CINT92 (6 integer programs) and
  - SPEC CFP92 (14 floating-point programs).
- 1995: a revision to the benchmarks (SPEC CPU95)
  - SPEC CINT95 (8 integer programs) and
  - SPEC CFP95 (10 floating-point programs).
- 2000: a revision to the benchmarks (SPEC CPU2000)
  - SPEC CINT2000 (12 integer programs) and
  - SPEC CFP2000 (14 floating-point programs).
- 2006: a revision to the benchmarks (SPEC CPU2006)
  - SPEC CINT2006 (12 integer programs) and
  - SPEC CFP2006 (17 floating-point programs).
- 2017: a revision to the benchmarks (SPEC CPU2017)
  - SPEC speed 2017 Integers (10 integer programs),
  - SPEC speed 2017 Floating Point (10 floating-point programs),
  - SPEC rate 2017 Integers (10 integer programs), and
  - SPEC rate 2017 Floating Point (13 floating-point programs).

The current release of SPEC's popular processor performance tests

- The complexity of applications grows with each new set of SPEC benchmarks to reflect the increasing demand on computers.

SPEC Methodology

- The SPEC methodology is to measure the time required to execute each program in the test suite; for example, the times might be T_{p1}, T_{p2}, T_{p3},… where the subscripts p1, p2, etc refer to the components of the suite.
- The SPEC suite is also executed on a so-called *standard basis machine* to give the values B_{p1}, B_{p2}, B_{p3},…
- The measured times are divided by the reference times to give the values T_{p1}/B_{p1}, T_{p2}/B_{p2}, T_{p3}/B_{p3},…
  This step ***normalizes*** the execution times of the components of the SPEC suite.
- Finally, *the normalized times are* ***averaged*** to generate a final SPEC metric for the machine being tested.

© 2014 Cengage Learning Engineering. All Rights Reserved.

SPEC Methodology

- Suppose that a reference machine takes 10, 100, and 5 seconds to perform tasks A, B, and C, respectively.
  Now, imagine that three test machines M1, M2, and M3 are tested to give the results specified in Table 6.14.
- Machine M2 has the best execution (***total***) time.

TABLE 6.11 Test Results for Three Hypothetical Machines

	Task A	Task B	Task C	Total
Reference machine	10	100	5	115
Machine M1	10	200	5	215
Machine M2	20	100	5	125
Machine M3	20	100	20	140

© Cengage Learning 2014

© 2014 Cengage Learning Engineering. All Rights Reserved.

SPEC Methodology

- Table 6.15 gives the same results normalized to the reference machine.
- As you can see, test machine M3 has the worst case execution time.

TABLE 6.11 Test Results for Three Hypothetical Machines

	Task A	Task B	Task C	Total
Reference machine	10	100	5	115
Machine M1	10	200	5	215
Machine M2	20	100	5	125
Machine M3	20	100	20	140

© Cengage Learning 2014

TABLE 6.12 Test Results for Three Hypothetical Machines after Normalization

	Task A	Task B	Task C	Total
Machine M1	1	2	1	4
Machine M2	2	1	1	4
Machine M3	2	1	4	7

© Cengage Learning 2014

© 2014 Cengage Learning Engineering. All Rights Reserved.

SPEC Methodology

- The SPEC benchmarks are sometimes said to give *unitless* values because the actual measurements for a given machine are divided by the same measurements for the reference or baseline machine.
  - For example, if the test machine gives the values 1s 10s 10s and the reference machine gives 2s 10s 20s, the normalized unitless results are ½ 1 ½.

© 2014 Cengage Learning Engineering. All Rights Reserved.

SPEC Methodology

- Note that: the individual SPEC benchmarks are averaged by taking the ***geometric*** *mean*, rather than the ***arithmetic*** mean.

- The ***arithmetic*** mean is calculated by adding together n values and then dividing the total by n
  - the ***arithmetic*** mean of the values 4, 5, 6 is $\frac{4+5+6}{3} = 5$

The result is not correct in page 384.

- The ***geometric*** mean is calculated by multiplying together n values and then taking the n^{th} root
  - the ***geometric*** mean of the values 4, 5, 6 is $\sqrt[3]{4 \times 5 \times 6} = 4.932$.

The result is not correct in page 384.

© 2014 Cengage Learning Engineering. All Rights Reserved.

SPEC Methodology

- The SPEC benchmarks are sometimes called the *best of a bad lot* or the *best metrics in the absence of anything that is more successful*.
- A ***criticism*** of SPEC is that it is ***CPU intensive*** and ***doesn't test the computer system***.
  - In particular, the effect of ***memory systems*** is not fully taken into account.
  - Moreover, the SPEC tests are not necessarily representative of the type of workload found in a ***multitasking*** environment – although the situation appears to have improved with the introduction of SPEC CPU2006 and CPU2017
- The SPEC organization has been criticized for periodically changing its benchmarks.
  - You could say that this prevents all systems being compared against an agreed baseline.
  - Equally, you could say that the nature of computers is changing and the way in which they are used is changing, and therefore it is essential to update the basis for comparison.

© 2014 Cengage Learning Engineering. All Rights Reserved.

SPEC CPU2017 Benchmarks

- SPEC CPU2017 benchmarks are written in either C, C++, and FORTRAN.
- These benchmarks cover both commercial and scientific applications.
- For more information about these benchmark programs, visit: https://www.spec.org/cpu2017/Docs/overview.html#suites

© 2014 Cengage Learning Engineering. All Rights Reserved.


Thanks
For Listening ☺

ARE YOU INTERESTED IN *IMAGE COMPRESSION*?

CS4481b/CS9628b
Image compression

Second term on Wednesdays from 10:30 am to 1:30 pm

NO final EXAM !!

Next year (Winter 2020)
Likely to be at the same date/time

To provide students with a solid understanding of the fundamentals and principles of various digital Image Compression Schemes

All required image-related background will be built during the course