16.02.2023 Thursday

Date: / / Page no:

COMPUTER PERFORMANCE

Perform. ∝ cost

cost
Performance
Trade off
cost ↑ Perform ↑

Quantitative (Measure) analysis
[Matrix are required]

For measuring the performance of any comp. on the basis of its architectural design, quantitative measures are required —

1. Time
2. Throughput

Job → program or many jobs in single prog.

(I) Time (unit : s) min.

(i) Execution Time — CPU bound (CPU is required) [majority] — IO bound instruction (I/p & O/p are required)

(ii) Response Time

(iii) Latency

ET: Time required by CPU to execute a job. T(OS) → not considered T(CPU)

$$T(CPU) + T(I/O) = ET$$

RT: context switching b/w CPU & I/O change. to execute a job.

$$T(CPU) + T(I/O) + T(extra) = RT$$

T(extra): waiting time [some device is busy], fetching time, context switching [CPU ↔ I/O] performed by OS [group of instr. to maintain sys]

User initiates cmd → terminates : RT

L: → Latency of CPU → T(CPU) to do job
→ L(Memory) → T(memory) to do job → READ / WRITE
→ L(Instruction) → Time required to execute a instruction

Consider avg. access time of memory per inst.

Unit of all → (s, min, hours)

(II) Throughput (unit : per second) · per minute

Rate at which job is executed.

ex. : 2 job per min.

TP: Throughput

| Performance $\propto \frac{1}{Time} \propto$ Throughput |

A B A is faster than B — P: Performance

$P(A) = n\,P(B)$

n>1 faster
n<1 slower
n=1 equal

[Performance(X) = P(X)]

$P(A) = n\,P(B)$

$$n = \frac{P(A)}{P(B)} = \frac{ET(B)}{ET(A)} = \frac{TP(A)}{TP(B)}$$

(1) A → 20 s
B → 25 s

$\frac{t_A}{t_B} = n = \frac{20^4}{25_5} = 0.8 = \frac{P_B}{P_A} \Rightarrow P_B = 0.8\,P_A$

P_B is slower

Ans: P_A is faster $\frac{5}{4}$ times than B

(2) Q)

Plane	Speed (mph) (miles per hour)	Range (miles)	Passenger capacity
Airbus-A	610	4150	470
Airbus-B	1350	4000	132

~~$t_A = 6.80$ h~~
$t_B = 2.96$ h

Job: miles

$\frac{P(B)}{P(A)} = \frac{TP(B)}{TP(A)} = 2.213$ ↙ see unit miles per hour

$P(B) = 2.2\,P(A)$ if only speed is considered

$$\frac{P(B)}{P(A)} = \frac{TP(B)}{TP(A)} = \frac{TP(B)}{TP(A)} \times \frac{TP(B)}{TP(A)} = \frac{610 \times 470}{1350 \times 132} = 1.60$$

(speed) (passengers)

Range: Fuel Refuelling Time (overhead)
~~Rep~~ (No Proper data)
But then too A > B (Range)

Compare the perf. for execution a job ~~for~~ in different scenario.
(RT, TP)

(1) upgrade a machine with new processor having faster clock.

Sol: faster CPU → faster processor
clk fasts → RT ↓ses (ET ↓ses)
↳ TP ↑ses

(2) multiprograming environment.

Job 1 Job 2 Job 3

content switching ↓ Busy

CPU

TP ↑ses
RT ↑ses

Waiting time of CPU ↓ No work for CPU ↓ses

Content Switching ↑ses.

T(waiting) ≥ T(switching)
↓ses ↑ses

∴ T ↓

(3) multiple computer (multiple jobs)

TP ↑ses
RT ↓ses
(each will do work individually)
~~Cost~~ ↑ses
cost vs perf. is not good
[Multiprograming is good]

(4) No. of processors are added.

TP ↑ses ↓ waiting time ↑se → if we ↑se job

Content Switching ~~↑ses~~ ↓ses

RT ↓ses.

~~CPU waiting time ↓~~ ~~TP ↑ses~~
~~RT~~ ~~Content Switching~~

CPI : cycles per instruction

Date: / / Page no:

execution time

CPU Time : Time required to execute a program.

Unit: program per unit time

multiple instructions per unit time

No of cycles required to per ~~instructions~~ program per unit time.

100 cycles (for Program) cycle time: 10 s duration of cycle

Time = 100 cycles/program × 10 s/cycle = 1000 s/program

CPU Time = No. of cycles × cycle Time

s/program Unit: cycles/program s/cycle

CPU time of program (Job) = instructions per program × cycles per instruction (CPI) → Standard term

If Rate is given

CPU Time = No of cycles / Rate → Clock Rate → Execution Rate

Clock Rate = 1 / Cycle Time

No. of instructions per program : ~~fixed~~ variable

No. of cycles per instructions : fixed.

Cycle Time : fixed.

Same type of operation

Then compare

Task: Program

instructions of same type

Time to execute a program / instruction → used to compare

18.02.2023 Friday

Date: / / Page no:

fetching time → depends on memory also along with CPU

In single operation if we can take program memory is 1 latency

In n operation → n latency

~~for~~ (CPU Time)$^{-1}$ = Throughput

Unit = programs/time

Cycle ? → Basis of clock inside computer.

CPU Time = No. of instructions per program × cycles per instruction × cycle time

Hardware dependent

⇒ CPU Time = Instruction Count × CPI × Cycle Time

Duration of 1 cycle.

Architecture dependent

If cycle Time ↓ CPI ↑ for same Instruction count and vice versa

Cycle Time $\propto \frac{1}{CPI}$ $\propto \frac{1}{\text{Clock Rate}}$

Can Instruction count be changed?

Yes, if comp supports diff. instruction set architecture.

1 add^r 3 add^r

For Job: N(Instr(1 add^r)) > N(Instr(3 add^r))

For 1 instruction Time(1 add^r) < Time(3 add^r)

Performance $\propto \frac{1}{\text{CPU Time}} \propto$ Throughput

Date: / / Page no:

$$\text{Average CPI} = \sum_{i=1}^{n} (CPI)_{i^{th}\ instr} \times P(i^{th})$$

CPI of i^{th} instruction; Probability of i^{th} instruction.

Q)

Operation	F_i	CPI_i
ALU	50%	1
Load	20%	5
Store	10%	3
Branch	20%	2

$F_i \times CPI_i$

2.2

Q) Clock rate 50MHz find execution time for 1000 instructions if CPI for a program 3.5

Sol.

CPU Time / Execution Time $= \frac{1000 \times 3.5}{50 \times 10^6}$ $M = 10^6$

$= 70 \times 10^{-6} = 70\ \mu s$

Q) If clock rate ↑ses 50 MHz → 100 MHz & other remains same.

Sol. Performance ∝ Clock Rate

Clock Rate doubles ⇒ Performance doubles.

$$\text{CPU Time} = \frac{\text{instruction count} \times CPI}{\text{Clock Rate}}$$

$$\text{Performance} = \frac{\text{Clock Rate}}{\text{instruction count} \times CPI}$$

Q) A comp. with 400 MHz clk. CPU executes a program in 10 s. To execute a same program in 6 s what should be the clk rate. Not that any increase in clk ~~rate requires~~ 1.2 times higher CPI than older one

Sol.

Date: / / Page no:

$$10 = \frac{\text{inst}^n\ ct. \times \alpha}{400}$$

$$6 = \frac{\alpha\ 1.2\ y}{(400 + y)}$$

$$\frac{10}{6} = \frac{10\,(400+y)}{2 \times 1.2\,y \cdot 400}$$

$800 = y + 400 \Rightarrow y = 400$

New Clock Rate $= 800$ MHz

inverse

IPC × Clock Rate → Instructions per second

CPI × cycle Time → second per instructions

instr. → cycle → sec

Execution Time ~~in of instructions~~ in sec.

Unit = mips → million instruction per second.

[Clock Rate in Mega Hz] mips higher → good machine

NOTE:

① Execution Time / Throughput is the only factor that can compare 2 systems

② No. of instruction count → can't compare, only ↓ ~~CPI should be given~~

③ No. of cycles on diff. machine Then it depends on clock rate.

22.02.2023
Wednesday.

Date: / / Page no:

(4) No. of cycles per second → Architecture depends
"No. of cycles"

(5) No. of cycles × cycle Time = constant
even though individually differ
Then ~~too~~ performance same

Q) 2 machine having same instruction set architecture

Machine A	Machine B
Clock cycle = 10 ns	20 ns
CPI = 2	1.2

Compare their performance

Sol. $\frac{(Perfor)_A}{(Perfor)_B} = \frac{(ipc)_A \times (ipc)}{(CR)_A}$: P

$P \propto \frac{1}{CPU\ Time} \propto \frac{1}{CPI \times Cycle\ time}$

$P_A = 1.2\ P_B$

Q) 2 compilers are tested for 100 MHz machines with three class of instruction A (1 cycle) B (2 cycle) C (3 cycle)
Compiler 1: 5M : A (No. of instructions : of type x)
1M : B 1M : C
Compiler 2: 10M : A 1M : B 1M : C
Calculate execution rate of 2 in mips

Sol. Execution Rate = IPC × Clock Rate
100 MHz
Average IPC
(3 types of instruction)

1: $CPI_A = 1$ (5 million) CPI_i

$(CPI)_1 = \frac{5}{7} \times 1 + \frac{1}{7} \times 2 + \frac{1}{7} \times 3$

$\frac{10 \times 1}{12} + \frac{1}{12} \times 2 + \frac{1}{12} \times 3$ $\frac{12}{15}$

$\Rightarrow ipc = \frac{7}{10}$ instruction per cycle

Mips are easy to compute but not true indicator

Date: / / Page no:

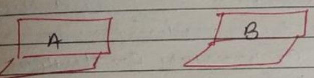
$(Execution\ rate)_A = (ipc)_A \times (clock\ rate)_A$
$= 0.7 \times 100 = 70$ mips.

$(ER)_B = (ipc)_B \times (clock\ rate)_B$
$= 0.8 \times 100 = 80$ mips

Not a true indicator

Pentium 200 MHz < Pentium pro 150 MHz
~~Pentium~~ Performance

Reason: memory op. takes less time I/O operation



Run same program
Less time
Better.

But problem. —
different types of activities.
graphics computer / Normal computer

Job. A Graphics type ~~G~~ faster | Slower.
[Unfair]

" A program which has all kind of operations
∴ we need "Computer Benchmark"

Standard programs to evaluate computer performance on different program.
[If developed by company then it can be biased for that company]

SPEC → develops computer benchmark. Set of programs

Comp. benchmarks are used to test the performance of a different machines

(2) Benchmark is a program or set of program used to evaluate the comp performance.

(3) It tests performance of a computer extensively not only to test some particular features.

Q) Who writes the Bench mark ?

(1) Small Benchmarks can be easily written by computer vendor (Designer) but they can be biased for their machine

(2) SPEC (System Performance Evaluation Cooperative)

It is consortium developed in mid of 90s to bring the common platform to test with respect of computer performance.

It's not only a single program but group of programs which are not written in any specific assembly language because in that case it can be biased.

(3) Comp. have agreed on a set of real programs & input.

(4) These programs are written in high level language. So it is a valuable indicator of H/w and S/w performance.

(5) Someone can design its machine instruction supported to benchmark programs and can still be abused since we are using set of programs so chances are less to abused.

Speed up of the System

Speed up = Older execution time of A / Newer execution time of A = Perf. new / Perf. old

↳ for single machine

Speed up thrice ⇒ execution time half

→ Speed up for the portion of program & not for whole

[others same]

New execution time = ET(new)

ET(new) = ET(old) × ((1 − Fraction Enhanced) + Fraction Enhanced / Speedup Enhanced)

1/2 + 1/2n ; 2/(1+n) ; speedup achieved for fractional part ; (1+n) − n ; (1+n)/(1+n)²

If speedup is achieved for 50% for program

then max Speedup = 2 (twice) for complete program (Fata SU enhanced = ∞)

Fractional Enhanced → major factor in increasing SU

on whole SU = SU enhanced

(3) A prog. runs in 100 s on a machine with multiply responsible for 80%. How much we have to improve the SU (multiplication) if we want the program to run 4 times faster

SU = 4 = 1 / ((1 − 0.8) + 0.8/SU_en) ; 0.8 = 4 − 0.2 ; SU_en = 0.8/0.21

16 time

$y = \frac{1}{0.2 + \frac{0.8}{x}}$ ⇒ $0.2 + \frac{0.8}{x} = 0.25$

$\frac{0.8}{x} = 0.05 \Rightarrow x = \frac{0.8}{0.05} = 16$

Q) How about making it 5 time faster

$0.2 + \frac{0.8}{x} = \frac{1}{5}$ ⇒ $x = \infty$

Q) Suppose we enhanced a machine making all floating pt. instruction runs 5 times faster. If the execution time for some benchmark before enhancement is 10s. What will be the speed up be if half of 10s is spend executing floating pt. instruction

$\text{Speed up} = \frac{1}{\frac{1}{2} + \frac{1}{2 \times 5}} = \frac{1}{\frac{1}{2}\left(\frac{6}{5}\right)} = \frac{1}{\frac{3}{5}} = \frac{5}{3}$

SU = 1.67

23.02.2023 Thursday

MICRO-OPERATION

Goto statement
↳ "interrupt"
(Change the execution chain)

↳ operations performed on the data stored in Registers.

CPU → ALU

Data is stored in Memory (Registers)
↓
Enumeratable type

(i) Register transfer | (ii) Arithmetic | (iii) Logical | (iv) Shift

$R_1 \leftarrow R_2$

transfer of content R_2 into R_1

Based → operator, operand

operator → Memory Reference, Register Refer, i/p o/p Reference

operand → 0 Addr, 1 Addr, 2 Addr

(i) Register transfer :-

$R_1 \leftarrow R_2$

The statement represents the transfer of the content of Register R_2 into Register R_1

◉ Control Functions

A statement that specify a register transfer implies that circuits are available from the o/p of the src register to the i/p of the dest. register.

interrupts → S/w interrupts (↳ Stack overflow, ↳ divide by 0); H/w interrupts (↳ Memory corrup, Shutdown)

Normally, we want the transfers only under a pre-determined control conditions. This can be shown by "if then" statement. (conditional operator)

eg. if (P = 1) then $R_1 \leftarrow R_2$

↳ Control function

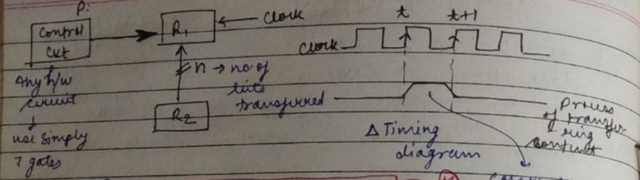
(normal if else type)

Symbolically $P : R_1 \leftarrow R_2$

P = boolean f^n

Bus designing [Bus → to reduce wire complexity to transfer data
↳ Data line
Address
Control & Selection

Date: / / Page no:



Any h/w circuit ↓ use simply 7 gates (AND, OR, NOT)

| ← transfer symbol |

If multiple statements depend on p

| $P: R_1 \leftarrow R_2 , R_3 \leftarrow R_4$ |

↳ comma → multiple statement

(H)				(L)			
7	6	5	4	3	2	1	0

High Low

R_1

| $R(L) = R(0-3) = R(1-4)$ |

8 bits

Half Low = 4 bits = 1 Nibble

(ii) Arithmetic Micro-operations.

In Register transfer micro operation does not change the information content when the binary information transfer from the src register to the dest. register. In other three types of micro operations change the information contents during the transfer.

| Microoperations are done sequentially once update we can't get back the data |

$R_1 \leftarrow R_1 + R_2$; $R_1 = 2$ $R_2 = 3$ $R_3 = 6$ // $R_1 = 5$

$R_1 \leftarrow R_1 - R_3$ // 5 − 6 = −1

updated as sequential stmt.

Date: / / Page no:

The basic Arithmetic micro operation are

(1) Addition

(2) Subtraction

(3) Increment & Decrement.

↳ 2 bit addition → Half Adder

3 bit " → Full Adder

(1) $R_1 \leftarrow R_1 + R_2$

(2) $R_1 \leftarrow R_1 - R_2$

How in comp.?

$R_1 \leftarrow R_1 + \overline{R_2} + 1$

2's complement

(3) $R_1 \leftarrow R_1 + 1$ (increment)

$R_1 \leftarrow R_1 - 1$ (decrement)

$R_1 + \bar{1} + 1$

$R1 + 0 + 1 = R_1 + 1$

↳ Take in bits

equal bits else it seems equal

R_1 − 0001

↓

4 bits

R_1 +

$R_1 \leftarrow 0101 = 5 = 0101$

$R_1 \leftarrow R_1 + 1$

+ 1 + 0001

6 = 0110   0110 = 6

R_1 0101 ≠ 5   0101 $R_1 \leftarrow R_1 - 1$

− 1   − 0001

0 1 0 1   0101

+ 0

X ✓ 0110

(iii) Logic Micro-operations:-

LMO specify binary operations for strings of bits stored in registers. These operations consider each bit of the register separately

$R_1 \leftarrow R_1 \vee R_2$ (OR) $R_1 \leftarrow R_1 \oplus R_2$ (XOR)

$R_1 \leftarrow R_1 \wedge R_2$ (AND) $R_1 \leftarrow R_1 \odot R_2$ (XNOR) opp. of XOR

$R_1 \leftarrow \overline{R_1}$ (NOT)

Operated bitwise without effecting neighbouring bit.

Some special LMO (Advance LMO)

(a) Selective set   (b) Selective complement

24.02.2023
Friday

(c) Selective Clear (d) mask
(e) Insert

Selective Set :- selective set operation set to 1 the bits in register A where there are corresponding one (1's) in register B

OR
1 0 1 0 Register A (before)
1 1 0 0 Register B
1 1 1 0 Register A (after)

A में 1 तो 1
पर A में 0 तो look B → 0 → 0, → 1 → 1

Selective Complement :- selective complement operation complements bits in A where there are corresponding 1's in B.

XOR
1 0 1 0 RA (before)
1 1 0 0 RB
0 1 1 0 RA (after)

] corresponding → 0 → same A, → 1 → complement A

(1)

(2) **Selective Clear :-** SC clear to 0 the bits in R_A only where there are corresponding 1's in B.

B में 1 → 0
1 0 1 0 R_A (before)
1 1 0 0 R_B
0 0 1 0 R_A (after)

] corresponding → 0 → same A, → 1 → 0

Mask operation :- MO is similar to the selective clear operation except that the bits of A are cleared only where there are corresponding '0s' in B

AND
1 0 1 0 R_A (before)
1 1 0 0 R_B
1 0 0 0 RA (after)

] corr. → 0 → 0, → 1 → same A

Insert Operation :- I op. inserts a new values into a group of bits. This is done by first masking the bits & then ORing them with the required values.
↳ 2 operations = Mask + ORing (Selective Set)

Overflow only happens Ashl

1 0 1 0 0 1 0 1
1 1 0 0 0 0 1 1 MASK
1 0 0 0 0 0 0 1
1 0 0 1 0 1 0 1 OR
1 0 0 1 0 1 0 1

RA : 1 0 1 0 0 0 1 0
0 0 0 0 1 1 1 1 Masking
RA : 0 0 0 0 0 0 1 0
1 1 1 1 0 0 0 0 ORing
RA : 1 1 1 1 0 0 1 0

Want to make 1 1 1 1 0 0 1 0

↳ जो Bits change करनी है वहाँ zero बाकी जगह 1
↳ जिन जगह changes दिए changes (same) else 0

(iv) Shift microoperation :
Shift Mop is used for serial transfer of data. The contents of a register can be shifted to the left or the right.
There are 3 types of shift Mop:-
(a) Logical (b) Circular (c) Arithmetic

(a) Logical
left & Right

$R_1 \leftarrow shl\ R_1$ — logical left shift

1 0 1 0 → discard, 0 1 0 0 insert
→ lsb पर 0 insert

If nothing is given shift by 1 bit only else do as it is

$R_1 \leftarrow shr\ R_1$

1 0 1 0 → insert 0, 0 1 0 1 (0 discard)

(b) Circular
cil, cir

$R_1 \leftarrow cil\ R_1$

1 1 1 0 → 1 1 0 1

$R_1 \leftarrow cir\ R_1$

1 1 1 0 → 0 1 1 1

(c) Arithmetic → same as shl but overflow
1st bit → signed bit if given 2's Comp.
1 bit: 1 → -ve value, 0 → +ve value

$R_1 \leftarrow Ashl\ R_1$
R_1 : 1 0 1 0 → -ve
discard 0 1 0 0 → +ve overflow cond^n nextpage
R_1 : 1 1 0 1 → -ve
discard 1 0 1 0 → -ve No overflow

Memory Refer

Date: / / Page no:

Overflow condⁿ

|Rn|Rn-1| . . . . . |R1|R0|

Same then no overflow → " Sign change after operations"

R1 ← 1010 → 1101 dis

→ only happens in Ashl

R1 ← Ashr R1

Arithmetic

1 0 1 0 → 1 1 0 1 discard

↓ keep MSB (Signed bit as it is) others shift.

∴ In this no overflow condⁿ

(1) NOTE: Ashl multiplies a signed binary number by 2

Ashr divides a — " — by 2

(2)