Computer Architecture and Assembly Language

Practical Session 12

Memory access
Performance

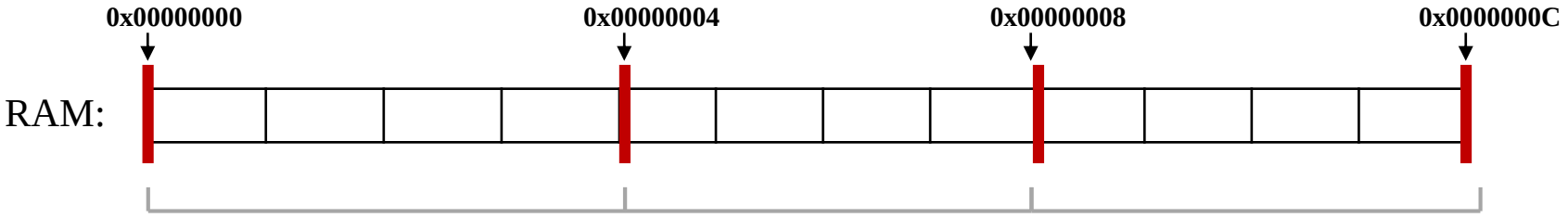
Memory access

הנחות יסוד:

- גישות לזיכרון מתבצעות ביחידות של 4 בתים
- החל מכתובת המתחלקת ב-4

instruction length = opcode length + immediate arguments length

- immediate נשמר או בבית 1, או ב 2 בתים , או ב 4 בתים



גישות לזיכרון:

1) Fetch – קריאת קוד הפקודה
2) Read – קריאת אופרנדים לחישוב
3) Write – כתיבת תוצאת החישוב

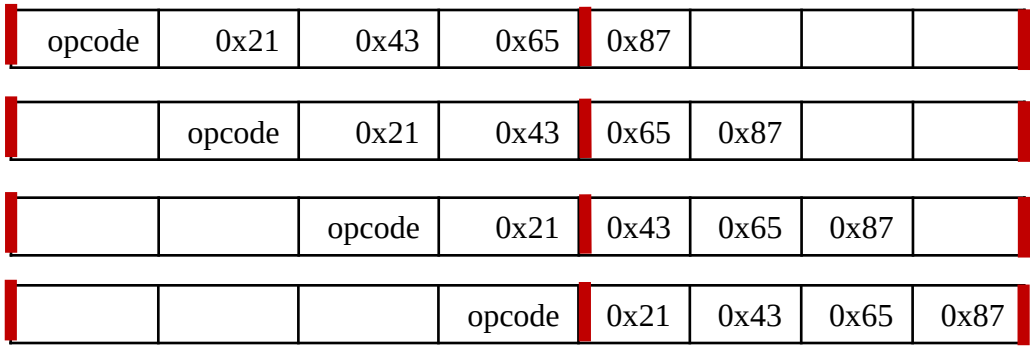
שאלה 1

כמה גישות לזיכרון, במקרה הגרוע, דרושות לביצוע הפקודה: MOV EAX, [0x87654321]
אם ידוע כי אורך ה-op-code הוא בית אחד?

פתרון:

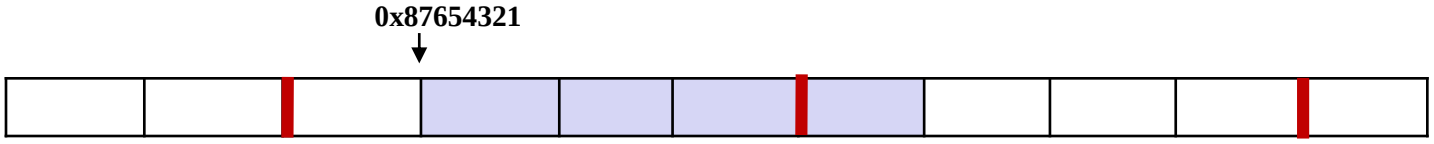
- Fetch – בתוך ה-op-code מצוין כל המידע הדרוש, פרט לכתובת. לכן, אורך הפקודה כולה 5 בתים (4 בתים נדרשים לקידוד הכתובת). במקרה הגרוע, קריאה של 5 בתים מהזיכרון דורשת **2 גישות לזכרון.**

RAM (.text):



- Read – יש לקרוא 4 בתים (התוכן הנמצא בכתובת הנתונה) ובמקרה הגרוע פעולה זו דורשת **2 גישות לזיכרון.**

RAM:



- Write – הכתיבה היא לתוך אוגר (אין גישה לזיכרון).

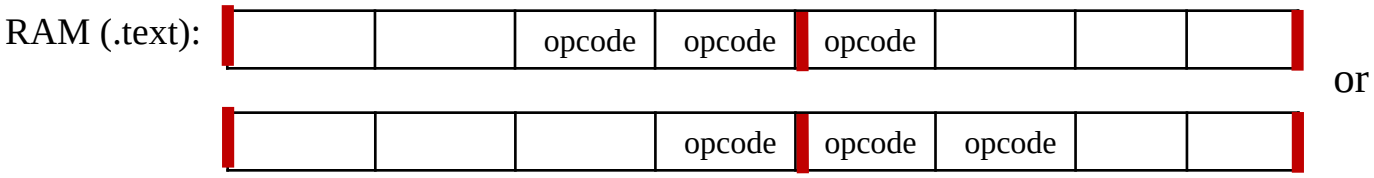
בסה"כ 4 גישות לזיכרון.

שאלה 2

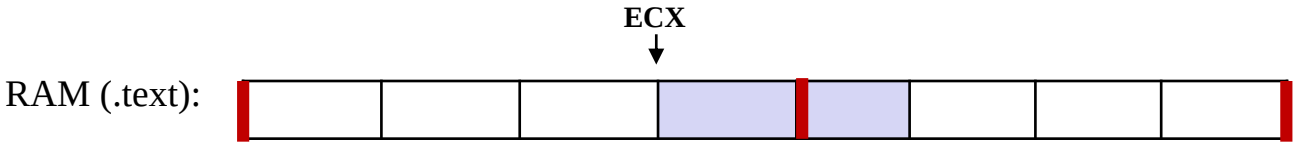
כמה גישות לזיכרון, במקרה הגרוע, דרושות לביצוע הפקודה: push word [ECX] אם ידוע כי אורך ה-op-code הוא 3 בתים ?

פתרון:

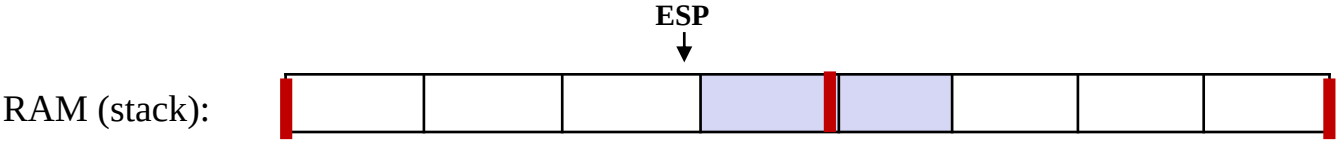
- Fetch – בתוך ה-op-code מצוין כל המידע הדרוש. במקרה הגרוע, קריאה של 3 בתים מהזיכרון דורשת **2 גישות לזיכרון.**



- Read – יש לקרוא 2 בתים (מילה, החל מהכתובת ב-ECX) ובמקרה הגרוע פעולה זו דורשת **2 גישות לזיכרון.**



- Write – הכתיבה היא למחסנית (זיכרון), ובמקרה הגרוע בו ESP מצביע לכתובת אי-זוגית, גם כאן נדרשות **2 גישות לזיכרון**.



בסה"כ 6 גישות לזיכרון.

שאלה 3

נתונה פקודה בשפת אסמבלי ואותה פקודה לאחר הקימפול בשפת מכונה:

ADD EBX, dword [label] 0x031D04030201

כמה גישות לזיכרון, במקרה הגרוע, דרושות לביצוע הפקודה?

פתרון:

- Fetch – אורך הקוד הבינארי 6 בתים. קריאתם מהזיכרון דורשת **3 גישות לזיכרון** לכל היותר
- Read – יש לקרוא 4 בתים החל מהכתובת 0x01020304 --> **גישה אחת לזיכרון**
- Write – הכתיבה היא לרגיסטר

⇦ בסה"כ 4 גישות לזיכרון

שאלה 4

כמה גישות לזיכרון, במקרה הגרוע, דרושות לביצוע הפקודה ADD dword [x], 0x10001 אם ידוע כי אורך ה-op-code הוא 2 בתים ?

פתרון:

- Fetch – אורך הקוד הבינארי 10 בתים – 2 עבור opcode, 4 עבור הכתובת x, ועוד 4 עבור מהספר המיידי. קריאתם מהזיכרון דורשת **4 גישות לזיכרון** לכל היותר
- Read – יש לקרוא 4 בתים החל מהכתובת --> **2 גישות לזיכרון** לכל היותר
- Write – יש לכתוב 4 בתים , וזה דורש 2 **גישות לזכרון** במקרה הגרוע.

⇦ בסה"כ 8 גישות לזיכרון

Performance

source book slides

Clock cycle: time between 2 consequent (machine) clock ticks.

Instead of reporting execution time in seconds, we often use cycles.

Clock rate: cycles per second

- 1 Hz = 1 cycle/sec
- 1Mhz = 10^6 Hz

different machine instructions take different amount of clock cycles

Example:

Machine X has 200 Mhz clock rate
=> X's clock rate is 200 * 10^6 Hz
=> X produces 2*10^8 clock cycles per second
=> X's cycle (time) is 1/ 2*10^8 = 5 nanoseconds (nanosecond = 10^{-9} seconds)

CPI (cycles per instruction) - quantity of clock cycles needed to execute an instruction

Speedup: = old execution time / improved execution time

Performance of a program A on machine X: $P_x(A) = 1 / \text{execution time}_x$

Machine X is n time faster than machine Y → $P_x / P_y = n$

Question 1

Some program runs in 10 seconds on computer A, which has a 400 Mhz clock.
We built a new machine B, which runs in 600 Mhz, but this machine requires each instruction 1.2 times as many clock cycles as machine A.
How much time would it take machine B to execute the same program?

Solution:

400 Mhz = $4*10^8$ Hz
=> machine A provides $4*10^8$ cycles per second
program runs 10 seconds on machine A
=> program execution takes $4*10^9$ cycles

= > on machine B it would take $1.2 * 4*10^9 = 4.8 *10^9$ cycles
600 Mhz = $6*10^8$ Hz
=> machine B provides $6*10^8$ cycles per second
= > on machine B it would take $4.8 *10^9 / 6 *10^8$ Hz = **8 seconds**

Question 2

There are two different classes of instructions: A and B .
- machine A has a clock cycle time of 10 ns. (nanoseconds) and a CPI (cycles per instruction) of 2 for class A instruction, CPI of 3 for class B instructions.
- machine B has a clock cycle time of 20 ns. and a CPI of 1.25 for both instructions classes.

A given program consists of 50% class A instructions and 50% class B instructions. Which machine runs this program faster?

Solution:

machine A: spends 2 * 10 = 20 ns per A class instruction
machine A: spends 3 * 10 = 30 ns per B class instruction
machine B: spends 1.25 * 20 = 25 ns per both A and B classes instruction

Time per instruction on machine A: (20 + 30)/2 = 25 ns
Time per instruction on machine B: 25 ns
=> **the machines have same performance for the given program**

Question 3

There are three different classes of instructions: class A, B and C.
They require 1, 3 and 5 cycles respectively.
There are two code sequences:
- first code sequence: 1 instructions of class A, 2 of B, and 1 of C.
- second code sequence: 6 instructions of class A, 1 of B, and 1 of C.

1. Which sequence will be faster?
2. By how much?
3. What is the average CPI (cycles per instruction) for each sequence?

Solution:

first code: 1*1+2*3+1*5 = 12 cycles => CPI = 12 / (1+2+1) = 3
second code: 6*1+1*3+1*5 = 14 cycles => CPI = 14 / (6+1+1) = 1.75

1. first code is faster
2. by 14/12 (speedup)
3. CPI = 3 for first code, CPI = 1.75 for second code

Question 4

A program runs in 100 seconds, with multiply (instructions) responsible for 80 seconds of its time. How much do we have to improve the speed of multiplication if we want the program to run 4 times faster? How about making it 5 times faster?

Solution:

100 sec total time – 80 sec multiplications = 20 sec rest instructions

4 times faster => 100 / 4 = 25 sec
=> 25-20=5 sec for multiplications
=> 80 / 5 = **16 times multiplication instruction speed needed**

5 times faster => 100 / 5 = 20 sec
=> 20-20=0 sec for multiplications
=> **impossible** ☹

Question 5

Given a code runs for 20 sec.
A code consists of 70% floating-point instructions.

We improved floating-point instructions run 7 times faster, but caused rest of the instructions run double the time.
What will the speedup be?

Solution:

70% of 20 seconds = 14 seconds for floating point instructions
=> 6 sec for rest instructions

execution time after improvement = 6*2 sec+ 14 sec / 7 = 12+2 = 14 sec
=> **speedup = 20 / 14**