Examen CN1 - Partea II - 12.06.2017

Nume: ………………………………………. Grupa: …………….

Fie următorul cod în assembly:

mov eax, 0

mov ecx, $n

mov esi, 0

mov edi, 0

**process\_element**:

cmp ecx, 0

je pwp

mov al, byte [v + ecx - 1]

cmp al, byte [X]

jg add\_greater

jl add\_less

**continue\_loop**:

dec ecx

jmp process\_element

**add\_greater**:

add esi, eax

jmp continue\_loop

**add\_less**:

add edi, eax

jmp continue\_loop

**pwp**: ; Print esi, edi

Procesorul ce execută acest cod rulează la frecvența de **1Ghz**. Metoda de *branch-prediction* folosită de fiecare este 1-bit counter (taken/not taken), starea inițială fiind *not taken*. Predictorul este unul local, astfel încât există un state-machine pentru fiecare branch.

Stim numărul de cicli de ceas pentru fiecare tip de instrucțiune: operațiile cu date și memorie (**mov**) durează 2 cicli, operațiile aritmetice/logice (**cmp**, **add**, **dec**) durează 1 ciclu, operațiile de salt necondiționat (**jmp**) durează 10 cicli, operațiile de salt condiționat (**jg, jl, je**) durează 40 cicli (dacă nu a fost prezis corect) sau 10 cicli (dacă a fost prezis corect).

Considerăm **$n=7** (dword), **$X=2** (byte), iar v este adresa de început a vectorului de octeți (1, 3, 3, 3, 1, 1, 3). Se cer următoarele:

1. **(1p)** Ce valori se afișează in labelul **pwp**.
2. **(5p)** Să se evidențieze prin ce stări trec branch-predictor-ele semnalându-se și care branch-uri au fost prezise corect, respectiv incorect.
3. **(2p)** Să se calculeze timpul de rulare al programului.
4. **(2p)** Se face următorul experiment:

n = 10, v = (1, 4, 4, 2, 3, 4, 5, 1, 1, 1), X = 2 ⇒ un timp de rulare t1.

n = 10, v = (1, 1, 1, 1, 2, 3, 4, 4, 4, 5), X = 2 ⇒ un timp de rulare t2 (t2 mult mai mic decât t1).

Ce proprietate al celui de-al doilea vector a dus la scăderea timpului de execuție? Justificați acest speed-up.

# Rezolvare

TODO: Rezolvare

Examen CN1 - Partea II - 16.06.2017

Nume: ………………………………………. Grupa: …………….

Fie următorul cod în assembly

mov ax, $a

mov bx, $b

**\_loop**: cmp ax, bx

jg \_greater

jl \_less

je \_final

**\_greater**: sub ax, bx

jmp \_loop

**\_less**: sub bx, ax

jmp \_loop

**\_final**: ; se afiseaza conținutul lui ax

Avem 2 procesoare, fiecare rulând la frecvența de **1Ghz**, care execută acest cod, procesoarele A și B. Metoda de branch-prediction folosită de procesorul A este **1-bit counter** (taken/not taken), iar metoda folosită de procesorul B este **2-bit counter** (strongly taken/weakly taken/weakly not taken/strongly not taken). Ambele pornesc cu o abordare pesimistă, counter-ul/state-machine-ul având starea inițială not taken, pentru procesorul A, respectiv strongly not taken, pentru B. Ambele implementează un branch-predictor global, astfel încât există un singur counter/state-machine global, pentru toate branch-urile.

Se știu numărul de cicli de ceas pentru execuția fiecărui tip de instrucțiune: operațiile cu date și memorie (**mov**) durează 2 cicli, operațiile aritmetice/logice (**cmp**, **sub**) durează 1 ciclu, operațiile de salt necondiționat (**jmp**) durează 10 cicli, operațiile de salt condiționat (**jg, jl, je**) durează 40 cicli (dacă nu a fost prezis corect) sau 10 cicli (dacă a fost prezis corect).

Fie **$a = 26**, **$b = 32**. Se cer următoarele:

1. **(4p)** Să se evidențieze prin ce stări trec branch-predictor-ele celor 2 procesoare, semnalându-se și care branch-uri au fost prezise corect, respectiv incorect.
2. **(3p)** Să se calculeze timpul de rulare al programului pe fiecare dintre cele 2 procesoare.
3. **(3p)** FIe un procesor C, rulând la frecvența de **1Ghz**, care folosește metoda de branch-prediction **2-bit counter**, implementând însă un branch-predictor local (câte un counter/state-machine pentru fiecare branch). Să se calculeze timpul de rulare al programului pe procesorul C. Care este speed-up-ul față de A și B? Argumentați ce a determinat acest speed-up.

# Rezolvare

TODO: rezolvare