Elaborarea unei analize privind conceptul de „overflow” la nivelul operațiilor aritmetice, concentrându-se pe modul în care arch x86. reacţionează când apare în cazul fiecăreia dintre cele 4 operaţii de bază. Care sunt cazurile luate în considerare de limbajul de asamblare și cum reacționează microprocesorul la acestea? Prezentați pentru fiecare dintre cele 4 cazuri, 2 exemple de cod sursă adecvate care ilustrează apariția unor astfel de situații și, respectiv, lipsa acestora. Explicați care este reacția exactă a sistemului de calcul pentru fiecare caz prezentat și ce poate face programatorul după ce apare o astfel de situație. Cum poate programatorul să evite situațiile de depășire?

CONCEPTUL DE DEPASIRE (OVERFLOW)

La nivelul procesorului si al limbajului de asamblare o DEPASIRE este o conditie/situatie matematica ce exprima faptul ca

* rezultatul uoe *nu a incaput in spatial rezervat acestuia* (ex: w:b=w )
* rezultatul uoe *nu apartine intervalului de reprezentare admisibil* pe acea dimensiune de reprezentare ( [0, 2^n - 1] / [-2^(n-1), 2^(n-1)-1] )
* uoe este *un nonsens matematic in respectiva interpretare* (cu semn sau fara semn) si astfel nu poate fi acceptata drept o operatie matematica corecta (neg+neg = poz / poz+poz=neg)

**CF (Carry Flag)** este flagul de transport. Are valoarea 1 in cazul in care in cadrul uoe s-a efectuat transport in afara domeniului de reprezentare al rezultatului si valuarea 0 in caz contrar. *CF semnaleaza depasirea in cazul interpretarii FARA SEMN*.

**OF (Overflow Flag)** este flag pentru depasirea in cazul interpretarii CU SEMN. Daca rezultatul uoe in aceasta interpretare nu a incaput in spatiul rezervat operanzilor, atunci acesta ia valoarea 1, altfel va avea valoarea 0.

**OF** va fi setat la valoarea 1 daca pentru operatia de **adunare** ne aflam in urmatoarele situatii ce provoaca depasire in interpretarea cu semn: P + P = N sau N + N = P

Ex: **0**011 + mov bx, 3 **1**000 +

**0**101 mov ax, 5 **1**000

= **1**000 add bx, ax = **1** **0**000

In cazul **scaderii**avem tot doua cazuri, consecinte a celor doua reguli de depasire in cazul adunarii: N – P = P sau P – N = N

* semantic cele 4 situatii exprima imposibilitatea acceptarii matematice a acestor operatii

Ex: 98 - **0**110 0010- 62h - 98-

200 **1**100 1000 C8h -56

= 154 = **1**001 1010 =9Ah =-102

(fara semn OF = CF = 1) (cu semn OF = CF = 1)

Cele doua interpretari asociate scaderii sunt incorecte dpdv matematic => **mP 80x86** seteaza OF=CF=1

Operatia de **inmultire** nu furnizeaza depasire la nivelul arhitecturii 80x86, spatiul rezervat rezultatului fiind suficient pentru ambele interpretari. Totusi, pentru a nu ramane neutilizate flag-urile CF si OF, s-a luat decizia ca in cazul in care in cadrul operatiei de inmultire dimensiunea rezultatului este identica cu cea a operanzilor (b\*b=b / w\*w=w / d\*d=d) CF = OF = 0 („no multiplication overflow”), iar daca avem in mod real una dintre situatiile b\*b=w / w\*w=d / d\*d=q, atunci CF = OF = 1 („multiplication overflow”)

mov al, 3 mov al, 15

mov bl, 2 mov bl, 15

mul bl ; al = al \* bl (incape pe byte) mul bl ; rezultatul word

OF = 0 OF = 1

In cadrul **impartirii**, daca catul obtinut nu incape in spatiul rezervat ( byte pt w/b, word pt dw/w, doubleword pt qw/dw) atunci se la semnala situatie de depasire la impartire cu efectul *„Run-time error”* si cu emiterea din partea **sistemului de operare** a unuia dintre cele 3 mesaje echivalente: *„Divide overflow / Division by zero / Zero divide”*. Avand in vedere ca in cazul unei impartiri incorecte programul se opreste nu se cunoaste valoarea flag-urilor CF, OF.

mov ax, 15 mov ax, 15

mov bl, 3 mov dl, 0

div bl ; 5 – byte div dl

OF = neschimbat Run-time error Zero divide

Ambele flaguri (CF si OF) sunt necesare deoarece in cadrul operatiilor de adunare si scadere in baza 2 se efectueaza simultan si identic doua operatii in baza 10 (cate una pentru fiecare interpretare), iar cele doua flaguri diferite se ocupa simultan de una dintre interpretari ( CF – fara semn, OF – cu semn). Ca urmare, la adunare si scadere nu este necesar sa precizam anterior desfasurarii lor interpretarea pe care o dorim operanziilor si rezultatului. Este suficient sa decidem ulterior efectuarii operatiei ce interpretare alegem.

Spre deosebire de aceste doua operatii, care functioneaza identic in baza2, indiferent de interpretare, inmultirea si impartirea CU SEMN si FARA SEMN, functioneaza diferit in fiecare dintre interpretari. Astfel, este necesar sa precizam anterior cum dormi sa fie interpretati operanzii prin MUL / DIV (fara semn) sau IMUL / IDIV (cu semn).

Pentru a evita situatii de overflow programatorul este nevoit sa realizeze verificari suplimentare. In cazul adunarii sau scaderii, programatorul poate verifica in prealabil valorile operanzilor. Atat pentru inmultire cat si pentru impartire, se pot utiliza tipuri de date cu capacitate mai mare. De asemenea, in ceea ce priveste impartirea, programatorul poate evita impartirea la 0, verificand daca impartitorul este diferit de 0 inainte de efectuarea operatiei.