Lucrarea 1

IDENTIFICAREA CARACTERISTICILOR PROCESORULUI

Dezvoltarea continuă a arhitecturii Intel a adus cu sine necesitatea implementării unei modalităţi software de identificare a caracteristicilor procesoarelor din sistemele de calcul. Acest mecanism de identificare a evoluat odată cu arhitectura Intel după cum urmează:

1. Într-o primă fază, cei de la Intel au publicat *secvenţe de cod* cu ajutorul cărora puteau fi detectate diferenţe minore la nivel de arhitectură sau de implementare pentru a identifica generaţiile de procesoare.

2. Odată cu apariţia procesorului 386, Intel a implementat *semnătura de identificare* a procesorului, care conţinea familia din care acesta făcea parte, modelul şi versiunea software-ului, această din urmă informaţie fiind disponibilă doar după realizarea operaţiei de RESET.

3. În momentul de faţă, prin *execuţia instrucţiunii CPUID pe procesor*, se poate obţine atât semnătura, cât şi un număr semnificativ de caracteristici ale procesorului.

Întrucât procesoarele se află într-o evoluţie continuă (Figura 1), specificaţiile instrucţiunii CPUID sunt şi acestea într-o extindere continuă.

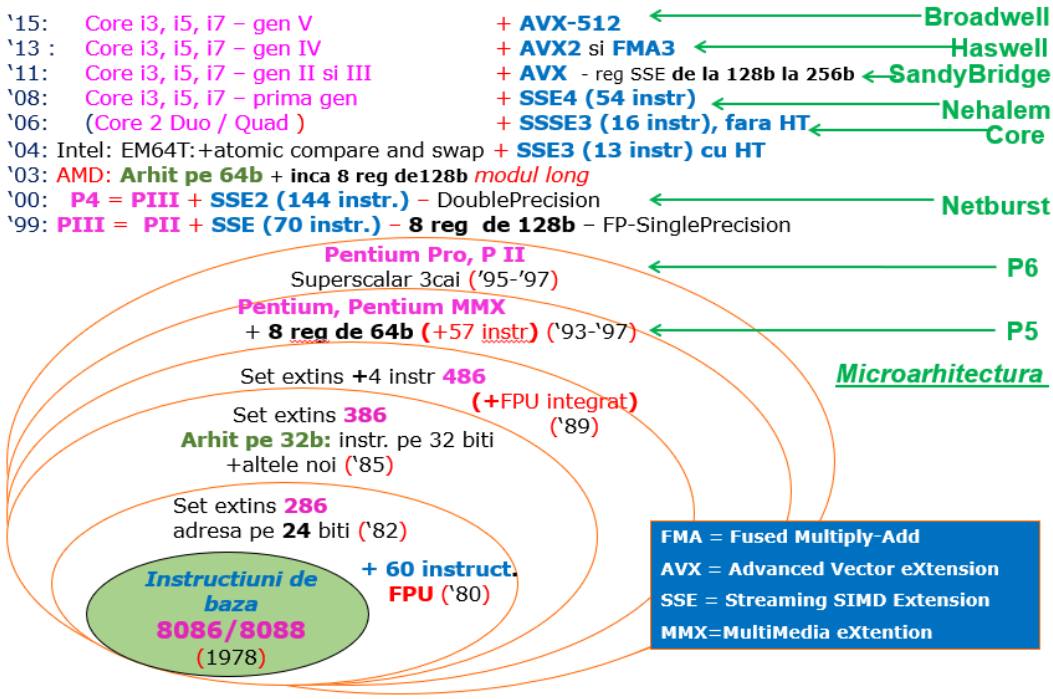


Figura 1. Evoluţia familiei x86: tipuri de microarhitectură şi seturi de instrucţiuni implementate

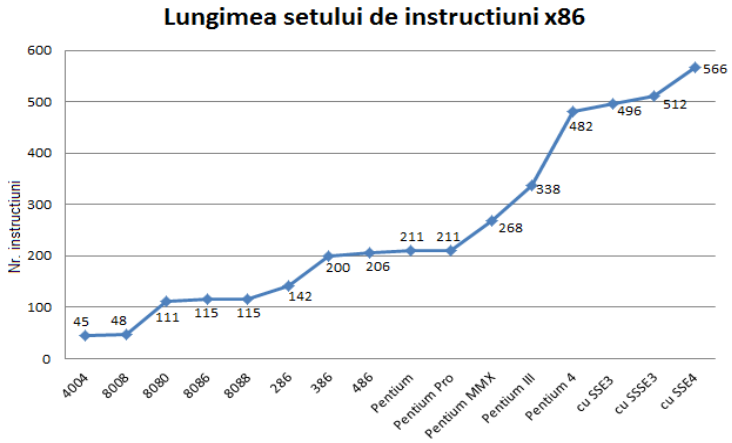


Figura 2. Numărul de instrucţiuni suportate de un procesor x86

Detectarea instrucţiunii CPUID

Instrucţiunea CPUID este disponibilă începând cu familia de procesoare Intel 486, drept urmare, în prealabil, trebuie determinat dacă procesorul suportă această instrucţiune. Acest lucru se poate face în două moduri:

1. Se execută instrucţiunea şi se verifică dacă apare o excepţie datorată unei instrucţiuni ilegale sau nu.
2. Se verifică dacă flagul ID (bitul 21 al registrului EFLAGS) poate fi (se lasă a fi) modificat prin program. În caz afirmativ, instrucţiunea CPUID este suportată.

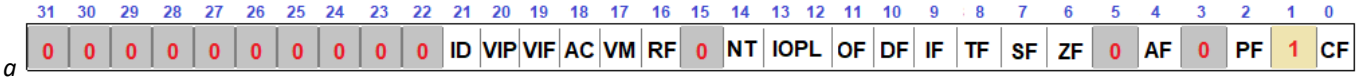


Figura 3. Ilustrarea conţinutului registrului EFLAGS

În Figura 3 este ilustrată structura registrului de flaguri (EFLAGS pe 32 de biţi), o descriere în detaliu a fiecărui flag fiind disponibilă în [1].

Secvenţa de testare a faptului că instrucţiunea CPUID este suportată pe procesor este redată în cele ce urmează. Mai multe exemple, pentru diferite modele de procesoare pot fi găsite în [2].

```
pushfd                  ; se salvează conținutul registrului EFLAGS pe stivă
pop eax                 ; se încarcă registrul EFLAGS în EAX
mov ebx, eax            ; se salvează EFLAGS în EBX (o copie a conținutului EFLAGS original)
xor eax, 00200000h      ; se setează flagul ID în reg. EAX
push eax                ; se salvează noua valoare din EAX pe stivă
popfd                   ; oare se va schimba valoarea în EFLAGS (????)
pushfd                  ; se salvează valoarea EFLAGS pe stivă, pt a verifica dacă s-a modificat EFLAGS
pop eax                 ; se încarcă valoarea EFLAGS în reg. EAX
cmp eax,ebx             ; se testează dacă s-a modificat bitul 21 în cadrul EFLAGS
jz NO_CPUID             ; dacă nu s-a modificat, instrucţiunea CPUID nu este suportată
```

Rezultatele instrucţiunii CPUID

Instrucţiunea CPUID suportă două seturi de funcţii:

- *setul de bază*, care furnizează informaţii uzuale ale procesoarelor x86 (asupra cărora producătorii s-au pus de acord)
- *setul extins*, care dă informaţii suplimentare, specifice fiecărui producător (Intel, AMD, etc.).

Exemplu:

Pentru EAX=1 -> cpuid oferă aceleaşi informaţii, adică semnătura CPU în EAX, resp. şirul Ascii al producătorului în EBX:EDX:ECX pentru ambele tipuri de CPU de referință, Intel şi AMD.

In schimb, pentru EAX=8000 0004h de exemplu, este posibil ca informaţiile oferite de Intel să nu fie aceleaşi cu cele oferite de o altă companie producătoare de CPU, de exemplu AMD.

Rezultatul instrucţiunii cpuid depinde de **conţinutul registrului EAX**, care funcţionează **ca un parametru de intrare pentru instrucţiune**. În funcţie de valoarea aflată în registrul EAX înainte de execuţia instrucţiunii, se obţin diferite ***informaţii de bază*** referitoare la microprocesor (vezi [2]).

Pentru a determina **cea mai mare valoare acceptată de instrucţiunea CPUID** (în sensul să nu interpretăm anumite informaţii care nu ar trebui interpretate pentru că nu au fost implementate pe acel procesor), trebuie înscrisă **valoarea „0”** în registrul EAX, urmată de apelarea instrucţiunii CPUID:

```
mov eax, 00h
cpuid           ; eax <- val max suportata la parametru pt cpuid pt aflarea informațiilor de bază
```

După execuţia secvenţei de instrucţiuni de mai sus, în EAX vom regăsi valoarea maximă care poate fi înscrisă în EAX ca şi parametru al instrucţiunii CPUID.

Pentru determinarea celei mai mari valori acceptată în registrul EAX de instrucţiunea CPUID care returnează ***informaţiile extinse despre procesor***, programul trebuie să înscrie în registrul EAX valoarea „80000000h” (deci b31=1) şi apoi să execute instrucţiunea CPUID:

```
mov eax, 80000000h
cpuid           ; eax <- val max suportata la parametru pt cpuid pt aflarea informațiilor de bază
```

Exemplu:

Dacă, de exemplu, pentru EAX=0, instrucţiunea cpuid va returna în EAX valoarea **0Ah**, atunci nu vom da parametrului EAX o valoare mai mare decât 0Ah la intrare, pentru a nu interpreta eronat informaţiile despre CPU nostru (valorile returnate de cpuid nu ar trebui interpretate pentru EAX**> 0Ah** la intrare).

Identificarea producătorului

Identificarea producătorului se face apelând instrucţiunea CPUID cu parametrul 0 în registrul EAX. Secvenţa de instrucţiuni returnează în registrul EAX valoarea maximă suportată (după cum am arătat mai sus), iar în regiştrii EBX, EDX şi ECX, şirul ASCII de identificare al producătorului (Figura 4).

	31...	23...	15...	7... 0
ECX	l (6Ch)	e (65h)	t (74h)	n (6eh)
EDX	I (49h)	e (65h)	n (6eh)	i (69h)
EBX	u (75h)	n (6eh)	e (65h)	G (47h)

Figura 4. Şirul ASCII de identificare al procesorului Intel

ID string	Producer
AuthenticAMD	AMD
CentaurHauls	Centaur
CyrixInstead	Cyrix
GenuineIntel	Intel
GenuineTMx86	Transmeta
Geode by NSC	National Semiconductor
NexGenDriven	NexGen
RiseRiseRise	Rise
SiS SiS SiS	SiS
UMC UMC UMC	UMC
VIA VIA VIA	VIA
Vortex86 SoC	Vortex

Tabel 1. Şirul ASCII pentru diferiţi producători

În cazul firmei Intel, şirul de identificare corespunzător este „**GenuineIntel**”. În [1] se găseşte o listă cu şirurile de identificare corespunzătoare altor producători, aşa cum arată Tabelul 1.

Semnătura procesorului

Semnătura procesorului este disponibilă începînd cu familia de procesoare 486. Aceasta este un număr pe 32 de biţi, format din 8 câmpuri, dintre care două sunt rezervate (Figura 5).

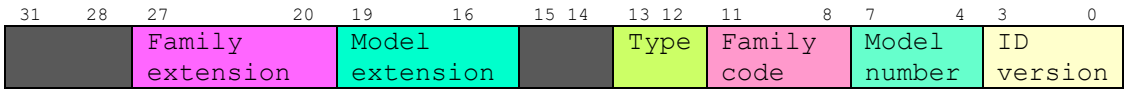


Figura 5. Structura semnăturii procesorului

Semnătura procesorului se poate obţine în două moduri: 1) după reset (în registrul EDX) sau 2) cu ajutorul instrucţiunii CPUID (doar pentru procesoarele care suportă instrucţiunea cpuid). În primul caz, semnătura procesorului se obţine în registrul EDX. În cel de-al doilea caz, registrul EAX trebuie să aibă valoarea 1 la apelul instrucţiunii CPUID, iar semnătura este returnată tot în registrul EAX. O descriere în detaliu a modului de interpretare a fiecărui câmp din semnătură se găseşte în [2] pentru procesoarele Intel, respectiv [3] pentru procesoarele AMD.

Numărul de serie al procesorului

Numărul de serie al procesorului este o valoare pe 96 de biţi, care poate fi folosită de aplicaţii pentru a identifica procesorul şi implicit sistemul în care acesta lucrează. Pentru a determina dacă această caracteristică este suportată, programul trebuie să apeleze instrucţiunea CPUID, cu registrul EAX având valoarea 1. Dacă bitul 18 din valoarea returnată în registrul EDX este 1, numărul de serie al procesorului este disponibil, în caz contrar acestă caracteristică nu este suportată sau este invalidată.

Numărul de serie al procesorului se obţine prin **concatenarea a trei entităţi de 32 de biţi**. Primii 32 de biţi se obţin în registrul EAX, prin apelarea instrucţiunii CPUID având 1 ca parametru. Această valoare trebuie salvată înainte de a apela din nou instrucţiunea CPUID cu EAX egal cu 3. După execuţia acestei instrucţiuni, registrul EDX conţine cei 32 de biţi din mijloc, iar registrul ECX, cei 32 de biţi de la sfârşitul numărului serial. Intel recomandă afişarea numărului de serie al procesorului ca şi 6 grupuri de 4 cifre hexazecimale.

Flagurile de caracteristici ale procesorului

Caracteristicile procesorului se obţin apelând instrucţiunea CPUID cu registrul EAX având valoarea 1. Dacă se doreşte obţinerea caracteristicilor extinse ale procesorului, registrul EAX trebuie setat la valoarea 80000001h, înainte de apelul CPUID. Caracteristicile sunt returnate în registrii ECX şi EDX, iar o interpretare a lor se găseşte în [2] şi [3].
Se va afla dacă PSN este validat sau nu şi dacă este suportat setul MMX.

```
Mov EAX, 1
Cpuid          ; MMX-> EDX b23 „MMX” =1 -> Da, este implementat MMX pe CPU respectiv
```

Identificarea mărcii

Odată cu apariţia modelelor 8 ale procesoarelor Pentium III, Pentium III Xeon şi Intel Celeron, Intel a extins conceptul de identificare prin adăugarea informaţiei de marcă (*Brand ID*). Aceasta este un număr pe 8 biţi care se obţine pe biţii 7 – 0 ai registrului EBX prin apelul instrucţiunii CPUID, având în EAX valoarea 1. Acest câmp a fost introdus pentru a elimina anumite ambiguităţi de identificare (cum era diferenţierea între Pentium II şi Pentium II Xeon – 512K L2 cache), oferind utilizatorului o valoare unică pentru fiecare marcă de procesor.

Istoria microprocesoarelor x86 - Arhitectura Intel

Compania Intel a produs numeroase cipuri şi arhitecturi, dintre care unele ajungând extrem de cunoscute. Primul cip produs de Intel a fost ***8086***, acesta fiind introdus pe piaţă în 1979. Intel 8086 a câştigat popularitate datorită faptului că IBM a decis să-l utilizeze în cadrul sistemului de tip PC (Personal Computer)[1]. Procesorul 8086 putea gestiona date pe 16 biți și putea lucra cu adrese pe 20 de biți (ceea ce însemna că în sistemul respectiv se putea utiliza o memorie de dimensiune 2^{20} octeţi, deci 1MB). Procesorul 8086 a fost împărțit în două părți: 1) unitatea de execuție (EU), care cuprindea regiștrii generali și 2) unitatea de interfață cu busurile (BIU), care includea coada de instrucțiuni, regiștrii segment și indicatorul instrucțiunii următoare (IP). Un cip foarte asemănător cu 8086, este ***8088*** care a fost utilizat în producţia de masă a computerelor personale pentru a reduce costurile de fabricaţie. Procesorul 8088 avea, la fel ca 8086, regiştrii interni de 16 biţi, deci unitatea de execuţie (EU) era identică. Deşi putea controla de asemenea 1MB de memorie, 8088 avea busul extern de date cu memoria, de doar 8 biţi în loc de 16 biţi (deci BIU era diferită), deoarece se dorea compatibilitate cu alte interfeţe de 8 biţi şi se urmărea în special reducerea costurilor de fabricaţie. Într-un astfel de sistem, cu bus de doar 8 biţi, dacă se dorea interfaţare cu o unitate pe 16 biţi, atunci aceasta ar fi luat 2 cicli procesor pentru a realiza acelaşi transfer care lui 8086 îi lua unul singur.

Aşa cum am studiat deja în semestrul anterior, procesorul 8086 avea 4 regiştri de 16 biţi, numiţi regiştrii de uz general: AX (acumulatorul primar), BX (registrul de bază folosit pentru adresare indexată), CX (registrul de numărare, contor) și DX (registrul de date). Fiecare dintre aceşti 4 regiştri a fost împărțit în două părți: jumătatea cea mai semnificativă a fost desemnată jumătatea "HIGH" sau superioară (desemnată de AH, BH, CH și DH), iar cea mai puțin semnificativă a fost desemnată jumătatea "LOW" sau inferioară (notată AL, BL, CL și DL). Pe lângă aceşti regiştri generali, 8086 mai avea 3 regiştri de tip pointer: pointerul de stivă (SP-stack pointer), care a fost folosit ca un offset la conţinutul din stivă, indicatorul de bază al stivei (BP- base pointer), care a fost utilizat pentru a indica baza stivei și indicatorul instrucțiunii următoare (IP-instruction pointer). Au existat, de asemenea, 2 regiştri index: registrul SI (index sursă), utilizat ca indicator al sursei pentru operațiile la nivel de șir și registrul DI (index destinație), utilizat ca indicator al destinației pentru operațiile la nivel de șir. Procesorul 8086 avea de asemenea un registru de indicatori de stare (FLAGS pe 16 biţi): biţii individuali din acest registru indicau diferite condiții, cum ar fi depășirea (la numere cu semn (OF) sau fără semn (CF)), semnul (SF), paritatea (PF), posibilitatea acceptării întreruperilor (IF) și așa mai departe.

Un program scris în limbaj de asamblare 8086 a fost împărțit în diferite segmente (blocuri speciale sau zone contigue din memorie) pentru a păstra anumite tipuri de informații (de cod, de date, de stivă). A existat un segment de cod (pentru păstrarea programului), un segment de date (pentru păstrarea datelor cu care lucra programul) și un segment de stivă (pentru păstrarea datelor din stiva programului). Pentru a accesa informațiile din oricare dintre aceste segmente, a fost necesar să se precizeze decalajul (offsetul) de adresă al acelei informaţii raportat la începutul segmentului corespunzător (adică se specifica adresa logică sub forma ***segment:offset***). Regiştrii segment erau folosiţi astfel pentru a reţine adresele de început ale segmentelor corespunzătoare din memorie (registrul CS păstra adresa de început a segmentului de cod, registrul DS păstra adresa de început a segmentului de date, iar registrul SS păstra adresa de început a segmentului de stivă). A existat, de asemenea, un al patrulea registru, numit registrul segmentului suplimentar (ES), acesta fiind utilizat de anumite operaţii specifice şirurilor pentru a gestiona adresarea memoriei. Datorită adresării segmentate, adresele au fost specificate pe 20 biţi, sub forma ***adresei logice***, ca *abcd:xyzu*, unde abcd a reprezentat valoarea din registrul segment și xyzu a fost offsetul raportat la adresa de început a acelui segment; fiecare dintre abcd şi xyzu au reprezentat valori pe 16 biţi.

În 1980, Intel a introdus cipul ***8087***, care a adăugat sistemului capabilitatea de a executa instrucțiuni în virgulă mobilă, dar şi o stivă de regiştri FPU de 80 de biți (a se vedea Figura 1).

---

[1] PC conţine de fapt un cip 8088 şi nu unul 8086, dar cele 2 sunt aproape identice, singura diferenţă fiind busul extern de date care la 8088 este de doar 8 biţi în loc de 16 biţi cât este la 8086

În 1982, Intel a implementat arhitectura ISA (Instruction Set Architecture) din 8086 asupra cipului ***80286*** (procesor pe 16 biţi cu bus de adresă de 24 biţi), iar ulterior, în 1985, asupra cipului ***80386*** (primul procesor pe 32 biţi, având bus de adresă de 32 biţi)- consultaţi Fig.1. Din punct de vedere al memoriei cu care putea lucra, procesorul 80286 putea adresa (datorită celor 24 biţi de adresă) până la 2^{24} octeţi, adică 16 MB, iar procesorul 80386 putea utiliza o memorie de 4 miliarde de octeți (mai exact 2^{32} octeţi). În timp ce cipul ***80287*** a reprezentat coprocesorul matematic pentru seria de procesoare Intel 80286, cipul ***80387***, lansat cu o întârziere de 2 ani faţă de cipul 80386, reprezenta perechea acestuia din urmă.

Tipul procesorului („pe n biţi") este dat de dimensiunea regiştrilor săi interni (de exemplu, 8086 avea regiştri interni AX, BX, ... pe 16 biţi). Procesorul 80386 a fost primul dintr-o familie de cip-uri numită adesea IA-32 (acronimul pentru Intel Architecture, 32-bit). Atunci când Intel a decis continuarea modelului 80286 de pe 16 biți și a lansat 80386 pe 32 de biți, inginerii proiectanţi au dorit ca aceste arhitecturi să fie compatibile înapoi (termenul utilizat fiind „***backward compatible***"), ceea ce înseamnă că programele scrise pentru un procesor mai vechi (şi deci mai puţin performant, de exemplu pe 8086) ar trebui să poată fi executate pe un procesor mai nou (şi deci mai performant de la vremea aceea, de exemplu 386). Prin urmare, Intel a păstrat aceeași arhitectură de bază și acelaşi set de regiştri, adăugând noile tehnologii peste aceasta de bază. (Noile caracteristici au fost adăugate la fiecare model succesiv, astfel încât compatibilitatea în avans nu a fost garantată.) Astfel, atunci când s-a trecut de la procesorul pe 16 biţi având regiştrii AX, BX, ... la procesorul pe 32 biţi (primul fiind 80386), denumirea regiştrilor a fost „moştenită", aceasta devenind **E**AX, **E**BX, ... unde prefixul „E" vine de la ***Extended***. În plus, aceşti regiştri extinşi încorporează în structura lor şi vechii regiştri AX, BX, aşa cum arată Figura 6. În cadrul programelor în limbaj de asamblare, programatorul are acces şi la regiştrii de 32 biţi (EAX, EBX, ...), dar şi la regiştrii AX, BX, sau chiar AH, AL, BH, BL, ... Reamintesc aici că nu toţi regiştrii aveau posibilitatea de a fi împărţiţi în parte high şi parte low (de exemplu regiştrii index SI, DI şi regiştrii pointer BP, SP sau IP).

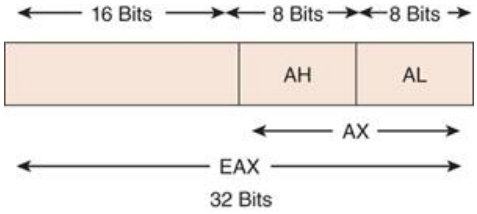


Figura 6. Registrul EAX, divizat în părţile componente de 16 biţi: *partea superioară* care nu se poate accesa întrucât nu are nume şi *partea inferioară* (AX), care la rândul ei se divide în AH şi AL (de câte 8 biţi fiecare)

După 4 ani de la lansarea cipului 80386, Intel a lansat prima linie de procesoare ***486***; acest nou tip de procesor era tot pe 32 biţi ca şi predecesorul lui, 80386, dar a primit acest nume (lipsit de prefixul 80), din cauza unei hotărâri judecătorești care a interzis înregistrarea ca marcă a unor numere (cum ar fi fost 80486). Ulterior, Intel a renunțat la denumirea pe bază de numere cu totul începând de la succesorul lui i486 – procesorul Pentium. Deşi atât 80386 cât şi 486 erau procesoare pe 32 biţi cu busuri de date de 32 biţi, din punct de vedere al performanţei, arhitectura i486 a adus o vastă îmbunătăţire faţă de 80386. Cipul 486 a fost primul cip care a inclus cache: 486 are (on-chip) ***cache de instrucţiuni şi de date (8KB cache de nivel L1)***, ***o unitate de tip floating-point (FPU) on-chip*** şi o unitate de interfaţă cu busurile (BIU) îmbunătăţită, care suporta o nouă tehnologie de la vremea aceea, pipeline. Datorită acestui design pipeline, secvențele de instrucțiuni simple s-ar putea executa pe durata unui singur ciclu de ceas (o instrucțiune se execută la fiecare bătaie a ceasului procesor, versus 2 cât erau necesare pe 386). Aceste îmbunătăţiri au condus la o dublare a performanţelor ALU faţă de un procesor 386 la aceeaşi rată de ceas: un 486 la 16 MHz a avut astfel o performanţă similară cu un 386 la 33 MHz. Un procesor 486 la 50 MHz executa, în medie, 40 MIPS (milioane de instrucţiuni pe secundă), dar ca vârf de performanţă, putea ajunge la 50 MIPS. Procesorul 486 conținea 1,2 milioane tranzistoare (în tehnologie 800 nanometri). [Wikipedia]

Procesorul 386 nu încorpora cache pe cip, dar putea să lucreze împreună cu cache off-chip (nu se considera a fi cache L2, ci tot cache L1, dar externă şi era mai lentă decât modelul încorporat de la 486). Deşi 486 a încorporat unitatea FPU, au mai fost fabricate cipuri ***'487*** pentru unele modele ***486***, dar ulterior, cipul ***80587*** (destinat utilizării împreună cu cipul NexGen's Nx586) a fost ultimul fabricat separat de CPU.

Procesoarele pe 16 biţi şi apoi cele pe 32 biţi au format așa numita familie x86 – cele pe 64 biţi care au urmat - cum ar veni ***x86 pe 64 biţi*** a primit numele de ***x64***; totuşi, în domeniul academic-educaţional încă se foloseşte termenul „x86 pe 64 biţi sau ***x86-64***" pentru a fi lucrurile mai clare celor care întâlnesc astfel de noţiuni pentru prima dată. Trecerea de la regiştrii de 32 biţi (EAX, EBX, ...) la regiştrii pe 64 biţi s-a realizat asemănător celei de la 16 biţi la 32 biţi, doar că noii regiştrii se numeau acum RAX, RBX, ... deci au primit prefixul R în loc de E.

Regiştrii cu regim special se consideră: regiştrii segment, [R/E]FLAGS şi [R/E]IP. Registrul [-/E/R] FLAGS este un registru de 16, 32 respectiv 64 biţi în care microprocesoarele din familia x86 păstrează starea curentă a procesorului. Regiştrii mai mari păstrează compatibilitatea cu predecesorii lor mai mici; biţii 15…0 ai registrului EFLAGS sunt de fapt biţii registrului FLAGS; similar, biţii 31…0 ai RFLAGS sunt de fapt biţii registrului EFLAGS. Partea superioară a registrului RFLAGS (în mod pe 64 biţi) este rezervată, iar partea inferioară este identică cu EFLAGS. Aici, la arhitectura de 64 biţi nu mai există noţiunea de segment.

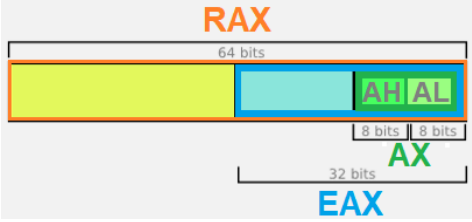


Figura 7. Registrul RAX, divizat în părţile componente de 32 biţi: *partea superioară* care nu se poate accesa întrucât nu are nume şi *partea inferioară* (EAX), care la rândul ei se divide în 2 părţi de câte 16 biţi: *o parte superioară fără nume* şi *o partea inferioară* reg. AX, divizat în AH şi AL (de câte 8 biţi fiecare)

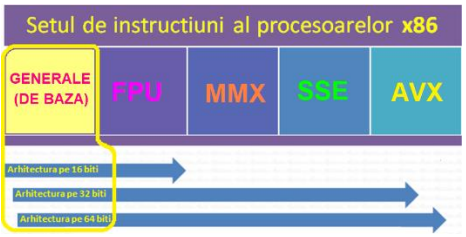


Figura 8. Evoluția setului de instrucțiuni

La arhitectura x86-64, pe lângă regiştrii „moşteniţi" de 64 biţi (precum RAX, RBX, RCX, RDX, RSI, RDI, RBP, RSP, RIP), mai există încă 8 regiştri de uz general de 64 biţi, denumiţi R8, R9, ..., R15. În modul pe 64 biţi, dacă se folosesc regiștrii R8,...,R15, aceștia pot fi adresaţi la nivel de **byte**, **word**, **doubleword** sau **quadword**, dar doar c.m.p.s. **8**, **16** sau **32** biţi pot constitui un **B**, un **W**, respectiv un **D**; de exemplu, octetul format de biții b15...b8 nu poate fi accesat direct, așa cum se putea face în cazul procesorului **8086** la *adresarea în mod real* (când se putea accesa și doar porţiunea AH din cadrul registrului AX);

Procesoarele x86, respectiv x64 conţin un set special de regiştri, care nu prea sunt folosiţi în programele uzuale; aceștia sunt regiştri de control (notaţi cu CR0,...,CR15), de depanare (DR0,...,DR7) și de test (TR0,...,TR7), mulţi dintre aceștia fiind rezervaţi. Dimensiunea acestor regiştri se consideră de 32 biţi când se lucrează în mod pe 32 biţi, respectiv de 64 biţi când se lucrează în mod pe 64 biţi. Acești regiştri nu vor fi abordaţi în exemplele prezentate aici. La arhitectura pe 64 biţi a fost introdusă şi noţiunea de ***double quadword*** ca o structură de 128 biţi.

Instrucțiunile ***SIMD*** (Single Instruction Multiple Data) pot creşte foarte mult performanța CPU la execuţie, aplicațiile tipice fiind procesarea semnalelor digitale și prelucrarea grafică (multimedia). Această categorie SIMD pentru procesoare din familia x86 şi x64 a fost divizată în ***mai multe seturi de instrucţiuni***, precum: MMX (1996), 3DNow! (1998), SSE (1999), SSE2 (2001), SSE3 (2004), SSSE3 (2006), SSE4 (2006), SSE5 (2007), AVX (2008), F16C (2009), XOP (2009), FMA4 (2011), FMA3 (2012), AVX2 (2013) AVX-512 (2015), unele fiind comune atât la procesoare Intel cât şi la AMD, altele fiind specifice doar la unul din cei doi producători (de exemplu 3DNow!, SSE5, F16C, XOP, apar la procesoare AMD). [2]

Primul pas în adăugarea acestor seturi de instrucţiuni pe CPU l-a realizat Intel prin introducerea în arhitectura IA-32 a setului MMX (multimedia extension). ***Setul MMX*** a fost însă ulterior perceput ca având două probleme:

1) refolosirea regiştrilor existenţi la acel moment pe CPU (regiştrii FPU), acest lucru împiedicând CPU să lucreze în acelaşi timp şi cu date FP şi cu date SIMD;
2) faptul că a funcţionat doar pentru numere întregi.

Astfel, în scurt timp, a apărut ***setul de instrucţiuni SSE*** cu suport pentru execuţia instrucţiunilor cu date FP pe un nou set de regiştri, independent: setul de regiştri **XMM**. În plus, setul SSE a mai adăugat şi câteva instrucţiuni cu numere întregi care foloseau regiştri MMX. SSE a continuat apoi cu SSE2, SSE3, SSSE3 (Supplemental SSE3) şi SSE4. Intel SSE4 s-a constituit dintr-un număr de 54 instrucţiuni, fiind compus de fapt din 2 subseturi: subsetul SSE4.1 (adică 47 instrucţiuni) şi subsetul SSE4.2 (încă 7 instrucţiuni, disponibile prima dată în procesoare Core i7, microarhitectură Nehalem). În mod diferit faţă de iteraţiile precedente ale SSE, setul SSE4 conţinea instrucţiuni pentru execuţia de operaţii care nu erau neapărat specifice aplicaţiilor multimedia, unele considerând registrul XMM0 ca operand implicit.

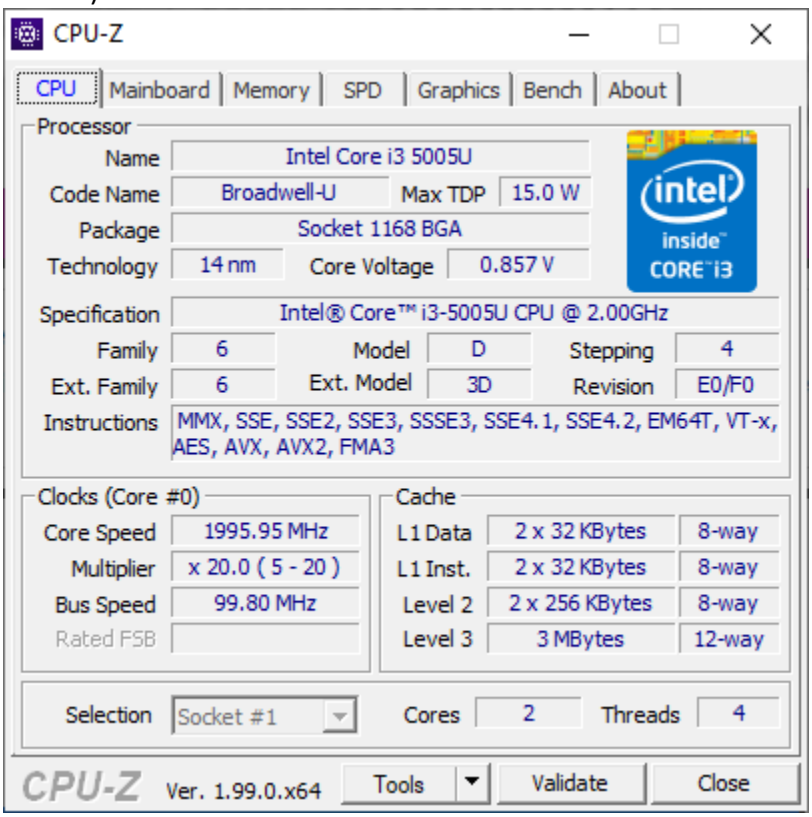
Termenul **Intel 64** (numit si **x64, IA-32e, EM64T, AMD64, x86-64**) este o **extensie a arhitecturii x86,** fiind compatibilă cu procesoarele anterioare din familia x86 (este de fapt trecerea de la 32 biţi la 64 biţi pentru seria de procesoare x86). A nu se confunda cu termenul **IA-64** care se refera la un alt fel de arhitectură (EPIC), dezvoltată de Intel şi HP (Hewlett Packard) împreună, implementată în procesoare Itanium si Itanium 2 dedicată serverelor şi staţiilor de lucru (preţ pe cip de aprox. 3000 $). Deşi cei de la Intel au fost cei care au gândit extinderea arhitecturală de la 16 biţi la 32 biţi, cei care au implementat extinderea de la 32 la 64 biţi au fost cei de la AMD: primul procesor compatibil x86 pe 64 biti a fost AMD- Athlon 64 (apărut în 2003) şi abia după un an au apărut primele procesoare Intel pe 64 biţi, din familia x86-64: **Xeon** si **Pentium 3.**

Bibliografie:

1. Eugen Lupu, *Sisteme cu microprocesoare – resurse hardware, prezentare, programare şi aplicaţii*, Editura Albastră, Cluj-Napoca, 2003.
2. *** - *Intel Processor Identification and the CPUID instruction*, Application Note 485, February 2005, disponibilă la https://www.scss.tcd.ie/Jeremy.Jones/CS4021/processor-identification-cpuid-instruction-note.pdf
3. *** - AMD Processor Recognition, Application Note, August 2004, disponibilă la https://www.amd.com/system/files/TechDocs/25481.pdf
[Wikipedia] https://en.wikibooks.org/wiki/X86_Assembly/X86_Architecture

Desfășurarea lucrării:

Tema 1. Se va rula aplicația „**CPU-Z**” de pe internet (https://www.cpuid.com/downloads/cpu-z/cpu-z_1.99-en.exe).



Tema 2. Se va rula aplicația „**Project1.exe**” din arhiva furnizată ca material de laborator.
Se va obține o imagine asemănătoare celei din Figura (cu space avansați printre informațiile oferite):

```
 eax in    eax      ebx      ecx      edx
00000000 00000014 756e6547 6c65746e 49656e69
00000001 000306d4 00100800 7ffafbbf bfebfbff
00000002 76036301 00f0b5ff 00000000 00c30000
00000003 00000000 00000000 00000000 00000000
00000004 00000000 00000000 00000000 00000000
00000005 00000040 00000040 00000003 11142120
00000006 00000075 00000002 00000009 00000000
00000007 00000000 00000000 00000000 00000000
00000008 00000000 00000000 00000000 00000000
00000009 00000000 00000000 00000000 00000000
0000000a 07300403 00000000 00000000 00000603
0000000b 00000000 00000000 0000000b 00000002
0000000c 00000000 00000000 00000000 00000000
0000000d 00000000 00000000 00000000 00000000
0000000e 00000000 00000000 00000000 00000000
0000000f 00000000 00000000 00000000 00000000
00000010 00000000 00000000 00000000 00000000
00000011 00000000 00000000 00000000 00000000
00000012 00000000 00000000 00000000 00000000
00000013 00000000 00000000 00000000 00000000
00000014 00000000 00000000 00000000 00000000
80000000 80000008 00000000 00000000 00000000
80000001 00000000 00000000 00000121 2c100000
80000002 65746e49 2952286c 726f4320 4d542865
80000003 33692029 3030352d 43205535 40205550
80000004 302e3220 7a484730 00000000 00000000
80000005 00000000 00000000 00000000 00000000
80000006 00000000 00000000 01006040 00000000
80000007 00000000 00000000 00000000 00000100
80000008 00003027 00000000 00000000 00000000

ID : "GenuineIntel"; CPUID level 20
```

```
Intel-specific functions:
Version 000306d4:
Type 0 - Original OEM
Familia 6 - Pentium Pro
Model 13 -
Stepping 4
Reserved 12
```

```
String extins producator: "Intel(R) Core(TM) i3-5005U CPU @ 2.00GHz"
CLFLUSH marimea liniei pt cache instructiuni : 8
Initial APIC ID: 2
Hyper threading siblings: 16

Feature flags bfebfbff:
FPU     Floating Point Unit
VME     Virtual 8086 Mode Enhancements
DE      Debugging Extensions
PSE     Page Size Extensions
TSC     Time Stamp Counter
MSR     Model Specific Registers
PAE     Physical Address Extension
MCE     Machine Check Exception
CX8     COMPXCHG8B Instruction
APIC    On-chip Advanced Programmable Interrupt Controller present and enabled
SEP     Fast System Call
MTRR    Memory Type Range Registers
PGE     PTE Global Flag
MCA     Machine Check Architecture
CMOV    Conditional Move and Compare Instructions
FGPAT   Page Attribute Table
PSE-36  36-bit Page Size Extension
CLFSH   CFLUSH instruction
DS      Debug store
ACPI    Thermal Monitor and Clock Ctrl
MMX     MMX instruction set
FXSR    Fast FP/MMX Streaming SIMD Extensions save/restore
SSE     Streaming SIMD Extensions instruction set
SSE2    SSE2 extensions
SS      Self Snoop
HT      Hyper Threading
TM      Thermal monitor
31      reserved
```

```
Informatii TLB si cache:
63: descriptor TLB/cache  necunoscut
03: Date TLB: pagina 4KB , 4 cai setate asociativ, 64 intrari
76: descriptor TLB/cache  necunoscut
ff: descriptor TLB/cache  necunoscut
b5: descriptor TLB/cache  necunoscut
f0: descriptor TLB/cache  necunoscut
c3: descriptor TLB/cache  necunoscut
Numar serial procesor serial: 0003-06D4-0000-0000-0000-0000
```

Tema 3: Se va rula sub Visual C++
Următorul program:
A) cpuid1.cpp:

```
// cpuid1.cpp : Defines the entry point for the console application.
//

#pragma hdrstop
#include "stdafx.h"
#include "stdio.h"
#include "stdlib.h"
#include "conio.h"
void decode_reg(int);
void decode_regWS(int);
void printreg(int reg);
```

```
void printregWS(int reg);
void PrintLevelCpuid(int level);
void cpuid(unsigned inp);
unsigned long extax,extbx,extcx,extdx;

void cpuid(unsigned inp)
{
   __asm
   {
      //.586
                mov eax,inp
                cpuid
                mov extax,eax
                mov extbx,ebx
                mov extcx,ecx
                mov extdx,edx
   };
}
int main()
{
  int i;
  unsigned long li,maxi,maxei;

  /* Print the information returned by CPUID for level 0 */
  cpuid(0);
  maxi=extax; //maximum parameter number
  for(i=0;i<=maxi;i++)
 {
  printf("\n\nCPUID level %d",i);
  PrintLevelCpuid(i);
  }
/*print the information returned by CPUID for level 0x80000000 */
  cpuid(0x80000000);
  maxei=extax; //maximum parameter number for extended functions

  for(li=0x80000000;li<=maxei;li++)
  {
  //printf("\n\nCPUID level %d",li);
  printf("\n\nCPUID level ");
  printregWS(li);
  PrintLevelCpuid(li);
  }
  if(maxei==0)
  printf("\n\nDoes not support extended level for CPUID ");
  printf("\n\n");

  /*Producer ID (Ascii string of the producer) and maximum level supported by CPUID*/
  cpuid(0);
  printf("ID : \"");
  for(i=0;i<4;i++) putchar(extbx >> (8*i));
  for(i=0;i<4;i++) putchar(extdx >> (8*i));
  for(i=0;i<4;i++) putchar(extcx >> (8*i));
  printf("\";");
  printf("\n");
  printf("CPUID level %ld\n",maxi);

  //wait for any key press
  getch();
  exit(0);
}

/* Register decoding x = register value */
```

```
void decode_reg(int x)
{
 x &= 0xff;
  printf("%02x ",x);
}
void decode_regWS(int x)
{
 x &= 0xff;
  printf("%02x",x);
}
//Print register value
void printreg(int reg)
{
                decode_reg(reg >> 24);
                decode_reg(reg >> 16);
                decode_reg(reg >> 8);
                decode_reg(reg );
}
void printregWS(int reg)
{
                decode_regWS(reg >> 24);
                decode_regWS(reg >> 16);
                decode_regWS(reg >> 8);
                decode_regWS(reg );
}
//Print cpuid level and returned values
void PrintLevelCpuid(int level)
{
    cpuid(level); // execute cpuid -> to load the EAX, EBX, ECX, EDX registers with desired values
  printf("\n    eax: ");
       printreg(extax);
  printf("\n    ebx: ");
       printreg(extbx);
  printf("\n    ecx: ");
       printreg(extcx);
  printf("\n    edx: ");
       printreg(extdx);
}
```

B) cpuid2.cpp:

```
#include "stdio.h"
#include "stdafx.h"
 int main (void)
{ int nr, neax0,nebx0,necx0,nedx0;
//printf ("Introduceti parametrul pt cpuid: \n");
//scanf ("%d ", &nr);
//switch to assembly
_asm
{ MOV EAX, 1
   cpuid
   mov neax0, eax
   mov nebx0, ebx
   mov necx0, ecx
   mov nedx0, edx
}
printf ("EAX: %02d \t",0);
printf ("EAX: %08X \t",neax0);
printf ("EBX: %08X \t",nebx0);
printf ("ECX: %08X \t",necx0);
printf ("EDX: %08X \n",nedx0);
return 0;
}
```

```
EAX: 00         EAX: 000306D4   EBX: 01100800   ECX: 7FFAFBBF   EDX: BFEBFBFF
Press any key to continue . . .
```

Tema 4. Se va rula folosind OllyDbg:

```
format db "valoarea lui EAX: %x", 0

mov EAX, 1
   cpuid
   ;printf();
   push EAX
   push dword format
   call [printf]
   add ESP, 4*2
```

!! nu uitati sa adaugati proiectului folosirea functiei printf

```
valoarea lui EAX: 306d4
Press any key to continue . . .
```