PM - CURS 2

x86 Registers:

- 8086 - primul proc. din fam. x86
  - ↳ are 14 regiștri pe 16-biți
  - ↳ fiecare GPR (reg. gen.) poate fi accesat pe octeți
  - ↳ pe sist. de 32-bit, reg. s-au extins la 32 și au primit extensia E (EAX ...) cu excepția celor de reg.
  - ↳ pe sist. de 64-bit, s-au mai adăugat 8 reg + extins reg. actuale

Floating-point registers:

↳ unitatea floating-point are propriii reg. și se află inclusă în procesor

↳ reg. au 80-biți (8) și 16-biți (3) -> moștenite de la coprocesorul matematic

Special registers: -

MMX:

- set de instr. SIMD
- oferă flexibilitate proc. ptr. procesarea aplicațiilor multimedia (ex: editare foto / video) unde este nevoie de o putere mare de proc. în paralel

3D Now:

- similar cu MMX (extinde MMX)

Streaming SIMD Extensions (SSE)

- extinde 3D Now

Physical Addr. Ext. (PAE):

- permite adresarea unei val. mai mare de 1GB ptr. proc.

Tick-Tock Model:

arhi: tectura

Net Burst:

- hyper threading = a avea 2 fire de exec. pe același core

Core Arch.:

.

Nehalem:

- proc. grafic ~~off-die~~ = ?

Sandy Bridge:

Atom:

- vers. slim-down de Nehalem

Haswell:

Skylake

Organizarea 8086:

Două componente:

- Execution unit
- BIU: - sum. permite efectuarea op. segmentate

↳ GPR:

↳ ALU: - doi operanzi (A, B)
- rezultatul (Y)
- F codifică op. pe care o poate face ALU
- permite efectuarea unor op. de bază logice + aritm.

↳ Flag Register

Codul mașină pt. x86:

opcode	Mode	operand 1	operand 2

↳ EU operation: Pași: - fetch
- decode
- efectuează op. (poate sau nu să folosească ALU)
ex: la adunare citește un reg. îl aduce la una din intrările ALU, la fel și pt. al doilea op., se efectuează op. și se duce rezultatul înapoi în reg.

↳ Pointers and Index Registers:

Le concatenează pînă de adr. și de date: inițial se consideră pînă ca fiind de adr., apoi se consideră ca biți de date.

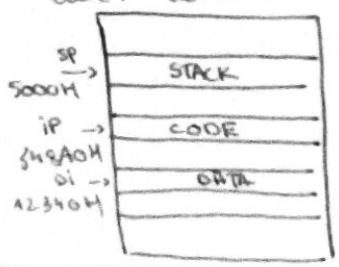
Generare adr. mem.: ?

Segmentarea mem.:
- reg. de reg. ne permite să avem un spațiu cu cod, un spațiu cu stivă, și unul cu date

Problema cu adr. segmentată este că trebuie să ne asigurăm că reg. sunt setați corect + putem avea două perechi care fac referire la aceleași reg.

Stocarea memoriei:
- byte-ul inf. este stocat la adr. inf.

Calc. adr. de mem.:



5000H = adr. de start pt. STACK
vezi

Fetching instrucțiuni:

1. CS + IP

2. IP ← IP + lung. instr. fetch-uite ca să pointeze la instr. urm.

Accesare mem. date.

↳ directă: MOV AL, [0300H]

↳ indirectă: MOV AL, [SI]

Locații rezervate de mem.:

↳ FFFF0H to FFFFFH - locații pt. codul de resetare a sist.

↳ 00000H to 003FFH - ~~tabelul~~ interrupt pointer table (256 de întreruperi pe 8 biți)

Întreruperile:

Execuția întrerupere:

1. Se oprește tot ce se execută și se salvează de unde ai plecat din cod = salvarea contextului

2. Dezactiv. întreruperilor pt. a nu avea întreruperi în întreruperi.

Dacă nu se dezactiv. => stack overflow

3. Sari la rutina de tratare a întreruperii (adr. este predefinită): La adr. aceea se găsește un jump care te duce la rutină

4. Se exec. rutina.

5. La fin. exec. există o instr. ?break? care face pop de pe stivă și se ajunge la adr. de unde am plecat în cod