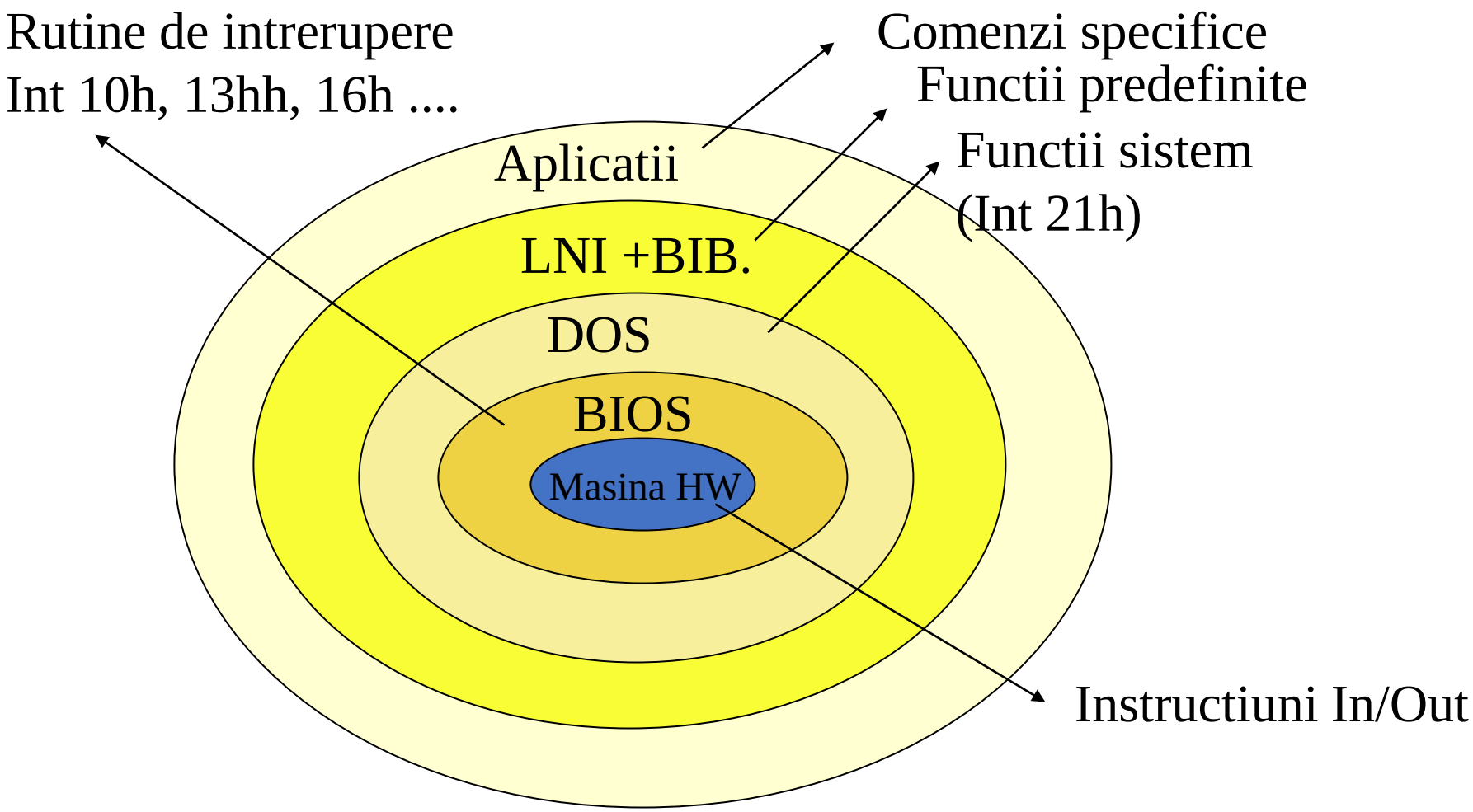
Accesul la resursele hardware

in modul real de functionare al procesorului

Resursele unui sistem de calcul

- Hardware
  - registre, indicatoare de conditie (flaguri)
  - memorie,
  - interfete si dispozitivede intrare/iesire,
  - sistemul de intreruperi,
  - ceas de timp real, alte timere
- Software
  - sistemul de fisiere
  - gestiunea taskurilor
  - gestiunea comunicatiei si servicii de sincronizare, etc.

Nivele access



Accesul la nivelul masinii hardware
exemplu: Tiparirea unui text la imprimanta

```
data     segment
text     db “hello world”
ltext    equ $-text
port     equ 378h
port_c   equ 37ah
port_s   equ 379h
strob    equ      00000001b
busy     equ      10000000b
ack      equ      01000000b
cmd_init equ  00000001b
data     ends
```

```
code segment
   assume cs: code, ds: data
start: mov ax, data
   mov ds, ax
   mov si, offset text
   mov cx, ltext
   mov dx, port_c
   mov al, cmd_init
   out dx, al
   mov dx, port_s
```

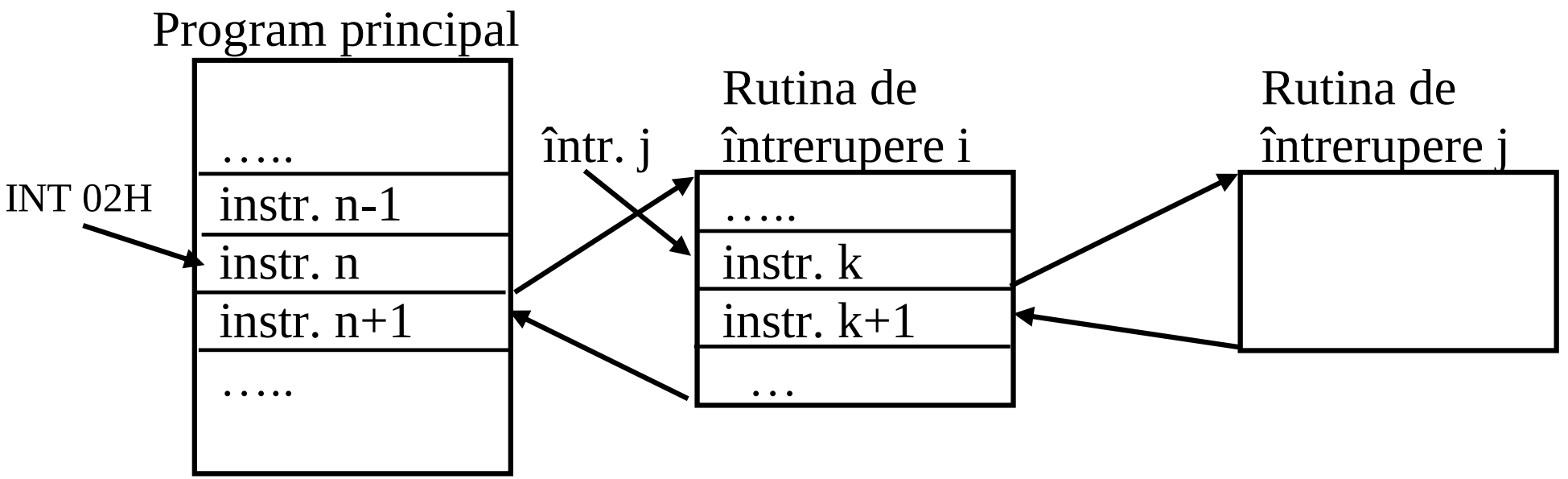
Tiparirea unui text la imprimanta (continuare)

```
et1:      in al, dx
   test al, busy    ; activ pe 1
   jz   et1
   mov al,[si]
   inc si
   mov dx, port
   out dx, al
   mov dx,port_c
   mov al, cmd_init
; se forteaza 0 pe strobe
   and al, NOT strob
   out dx, al
   mov dx, port_s
et2:     in al, dx
   test al, ack        ; activ pe 0
   jnz   et2
   mov dx, port_c
   mov al,cmd_init
; se forteaza 1 pe strobe
   or al, stb
   out dx, al
   mov dx, port_s
   loop et1
```

Tiparirea unui text la imprimanta (continuare)

```
    mov ax, 4c00h
    int 21h
code ends
    end start
```

Intreruperi



Accesul la resurse prin intreruperi BIOS

- intreruperi hardware - generate de interfete la aparitia unui eveniment (ex: receptia unui caracter, citirea unui sector, eroare de transmisie, etc.)
- intreruperi software – access la resurse prin rutine (functii) BIOS
  - fiecare tip de resursa (interfata) are rezervat un nivel de intrerupere
  - se apeleaza prin instructiunea INT n
- apelul functiilor:
  - se seteaza parametrii de apel in registrii procesorului
  - se executa instructiunea de intrerupere corespunzatoare
  - la revenirea din rutina se testeaza CF (0 - OK, 1-eroare)

- exemple de intreruperi BIOS (software)
  - INT 10h - functii de ecran
  - INT 13H - functii FDD, HDD
  - INT 16H - functii de tastatura
  - INT 14H - functii de comunicatie pe canalul serial
  - INT 17H - functii de tiparire

Accesul la resurse prin intreruperi BIOS
exemplu: citirea unui sector de pe disc

```
data segment
buf      db 512 dup(?)
data ends
code segment
   assume cs:code, ds: data
start: mov ax, data
   mov ds, ax
   mov es, ax
;initializare parametrii de apel
   mov ah, 02 ; functia de citire
   mov al, 1 ; nr. de sectoare
   mov cl, 5  ; nr sector
   mov ch, 0 ; nr. pista
   mov dl, 0 ; nr. disc
   mov dh, 0 ; nr. cap
   mov bx, offset buf
   int  13h
   jc err
   ....
err: ; cod pt. eroare
code ends
   end start
```

Accesul la resurse prin apeluri sistem (system calls)

- se folosesc concepte mai abstracte:
  - fisier - pt. accesul la dispozitivele de stocare
  - canal - pt. dispozitivele de transmisie a datelor
  - dispozitiv standard de intrare
  - dispozitiv standard de iesire
- modul de acces:
  - se seteaza anumiti parametri in registrele procesorului
  - se construiesc anumite structuri de date pt. acces (handler)
  - se apeleaza intreruperea INT 21h
  - se testeaza corectitudinea efectuarii operatiei
- Exemple
  - 01 - citirea tastaturii cu ecou
  - 02 - afisarea unui caracter
  - 05 - tiparirea unui caracter
  - 08 - citirea tastaturii fara ecou
  - 14h - citirea secventiala cu FCB
  - 15h - scrierea secventiala cu FCB
  - 3fh - citirea unui fisier
  - 40h - scrierea unui fisier
  - 31h - revenirea in procesul parinte cu pastrarea alocarii memoriei
  - 4ch - revenirea in sistemul de operare

Accesul la resurse prin apeluri sistem (system calls) exemplu:

```
; tiparirea unui caracter

    mov al, “x”
    mov ah, 05
    int 21h

;citirea unui caracter de la
; disp. de intrare standard

    mov ah, 01
    int 21h

; in al va fi caracterul citit
```

```
; iesirea dintr-un proces cu
;pastrarea alocarii memoriei

mov ah, 31h
mov al, cod_retur
mov dx, dimensiune
int 21h

;dimensiunea memoriei rezervate
   se da in paragrafe de 16 octeti
```

Executia periodica unor operatii (intreruperea 1CH)

- Implementare: prin ceasul de timp-real
  - contor (timer) care genereaza periodic intreruperi - la fiecare 18,2ms
  - rutina de tratare a intreruperii contine instructiunea INT 1CH
  - rutina de tratare a intreruperii 1CH contine o singura instructiune : IRET - revenire din rutina
  - prin redirectarea intreruperii 1CH catre o anumita rutina, aceasta va fi apelata la fiecare intrerupere a ceasului de timp real
  - rutina trebuie sa fie rezidenta in memorie

Optimizarea programelor

Cand si ce se optimizeaza

- regula 90/10 - 90% din timp se executa 10% din cod
  - consecinta - daca se elimina 90% din codul rar folosit imbunatatirea este de 10%
- ce se doreste?
  - timp redus de executie sau reducerea spatiului de memorie utilizat
- cum se masoara timpul processor ocupat de fiecare modul – profiler
- cand este bine sa se optimizeze:
  - de la inceput:
    - se optimizeaza si partea nesemnificativa
    - programul se scrie greu, se intelege si mai greu
  - la sfarsit:
    - prea tirziu

Optimizarea este necesara ?

- Contra-argumente:
  - viteza mare a procesoarelor, a memoriilor a magistralelor
  - spatii de memorie foarte mari
- Cand este necesara optimizarea:
  - prelucrari de date multimedia
  - prelucrari de semnale
  - sisteme de control in timp-real, sisteme reactive

Trei tipuri de optimizare

- Alt algoritm, mai bun
  - optimizare de nivel inalt
  - gasirea unui algoritm cu grad mai mic de complexitate $O(n^2)$ =>$O(nlg(n))$
- Algoritm implementat mai bine
  - optimizare de nivel mediu
- Micsorarea numarului de cicluri de ceas de executie
  - optimizare de nivel scazut

Tehnici de optimizare

- reducerea numarului de bucle imbricate
- reducerea timpului de executie al buclei interioare
- utilizarea poantorilor in adresarea elementelor unor structuri de date
- parcurgerea structurilor de date prin incrementarea si decrementarea poantorilor in locul calcularii adresei emenentului
  - ex: tab[i][j]

    $adr\_element_{ij}$ = adresa(tab)+i*lung_rand +j
- utilizarea registrelor interne ale procesorului in operatiile curente

Exemplu de optimizare

- Problema: filtrarea unei imagini de 100 de ori, rezolutie 256*256 pixeli
- Variante:
  - program Pascal - 45s
  - program C - 29s
  - asamblare (V1) - 6s
    - s-au folosit deplasamente precalculate in locul calculului de adresa prin indecsi
  - asamblare (V2) - 4s
    - s-au folosit registre interne in locul variabilelor de memorie

Exemplu de optimizare

- Variante (continuare):
  - asamblare (V3) 2,5s
    - s-a evitat recopierea imaginii intermediare in matricea initiala
  - asamblare (V4) 2,4s
    - s-a redus dimensiunea variabilelor de la intreg la caracter pt. a beneficia de mem. cache
  - asamblare (V5) 2,2s
    - s-a schimbat algoritmul de filtrare, s-au lutat in calcul numai 4 vecini ai pixelului in loc de 8 (rezultatul este diferit)

Apelul procedurilor scrise in asamblare din limbaje de nivel inalt

- Cand se justifica:
  - la accesul direct al unor resurse fizice (ex: interfete de I/E, memoria video, etc.)
  - pentru cresterea eficientei unor secvente critice
  - functiile de acces la resurse au fost scrise (deja) in asamblare
- Dificultati:
  - alta filozofie de scriere a programelor (ex: registre in loc de variabile, date simple in locul celor structurate)
  - transferul parametrilor de apel si a rezultatelor
- Cum:
  - Cod inline
  - Proceduri cu respectarea conventiilor de apel

Alte limbaje de asamblare

Arhitectura MIPS (million instructions per second)

- Arhitectura RISC Reduced Instruction Set Computer
- In contrast cu arhitectura CISC Complex Instruction Set Computer)
- J. Hennesy, 1981, arhitectura “academica” cu f. multe implementari practice (ex: PIC32, ARM, PlayStation)
- Caracteristici arhitecturale:
  - Set redus de instructiuni (aprox. 35)
  - Instructiuni de lungime fixa (32 biti)
  - Accesul la memorie numai prin 2 instructiuni Load/Store
  - Numar redus de moduri de adresare
  - Principiu RISC sacrifica totul pentru viteza => arhitectura simpla
  - ce permite executia instructiunilor intr un timp minim
  - Frecventa ceasului sistem este mai mare decat in cazul arhitecturilor CISC

32 de registre interne

- Banc de registre ce compenseaza partial lipsa instructiunilor cu memoria
- Adresarea registrelor:
  - Cu $n (n=0 31)
  - Cu $xn unde x=v,a,t,s,k,sp,gp,ra si n=0,1,2,…9
  - x indica functia indeplinita de registru
  - ex: t=reg. temporar; s= registre salvate la apelul de rutina
  - sp=stack pointer, gp=global pointer, v= valori generate in urma evaluarii unor expresii
  - Registrul $0 contine valoarea 0

Formatul instructiunilor

- Instructiunile au lungime fixa de 32b dar un continut variabil
- Instructiuni de tip “R” registru

  opcode (6b), rs (5b), rt (5b), rd (5b), shift (5b), functie (6b)
- <instr> rd, rs, rt
  - rd registru destinatie
  - rs registru sursa
  - rt registru tinta (target)
  - Ex: add $t1, $t2, $t3 ;$t1=$t2+$t3

Formatul instructiunilor

- Instructiune de tip “I” cu valoare imediata

  opcode (6b), rs (5b), rt (5b), Imm/Addr (16b),

- <instr> rt, rs, IMM
  - rs registru sursa
  - rt registru tinta (target)
- Ex: addi $t1, $t2, 55 ; $t1=$t2+55

- Instructiuni de tip “J” jump

  opcode (6b), Addr (26b),

- <instr> LABEL
- Ex: j et1 ;jump

Tipuri de instructiuni

- Aritmetice si logice
- Load si Store
- Salturi si ramificatii (branch)

Instructiuni aritmetice si logice

- add $rd, $rs, $rt ;$rd = $rs + $rt
- addi $rt, $rs, imm ;$rt = $rs + imm
- sub $rd, $rs, $rt ;$rd = $rs --$rt
- mult $rs, $rt ;$LO = $rs * $rt
- div $rs, $rt ;$LO=$rs/$rt; $HI=$rs % $rt
- and $rd, $rs, $rt ; $rd = $rs & $rt
- andi $rt, $rs, imm ; $rt = $rs & imm
- or $rd, $rs, $rt ; $rd = $rs | $rt
- ori $rt, $rs, imm ; $rt = $rs | imm

Instructiuni Load si Store

- Load word
  - lw $rt, offset($rs) ; $rt = MEM[$rs + offset]
- Load byte
  - lb $rt, offset($rs) ; $rt = MEM[$rs + offset]
- Store word
  - sw $rt, offset($rs) ; MEM[$rs + offset] = $rt
- Store byte
  - sb $t, offset($s) ;MEM[$s + offset] = (0xff & $t)

Instructiuni de salt

- Salturi neconditionate
  - j target ; PC = (PC & 0xf0000000) | (target << 2)
  - jr $rs ; salt cu registru PC = $rs;

- Salturi conditionate (ramificari branch)
  - Branch on equal
    - beq $rs, $rt, offset ; if $rs=$rt PC=PC+(offset<<2))
  - Branch on greater than or equal with zero
    - bgez $rs, offset ; if $rs >= 0 PC=Pc+ (offset << 2))

Comparatie cu Intel x86

Parametru	ISAx86	MIPS
Nr. instructiuni	Foarte multe (~150)	Putine (35)
Complexitate instr.	Instructiuni complexe	Instructiuni simple
Format instr.	Variabil 1-16 octeti	Fix 4 octeti
Instructiuni cu memoria	Majoritatea instructiunilor	Doar instr. Load si store
Moduri de adresare	Multiple, complexe	Putine, simple
Executia instructiunilor	Mai multe cicluri	Un ciclu
Numar registre	8 reg. generale	32 de registre
Arhitectura UCP	Complexa (CISC)	Simpla (RISC)
Programare	Simpla, flexibila	Rigida, complicata