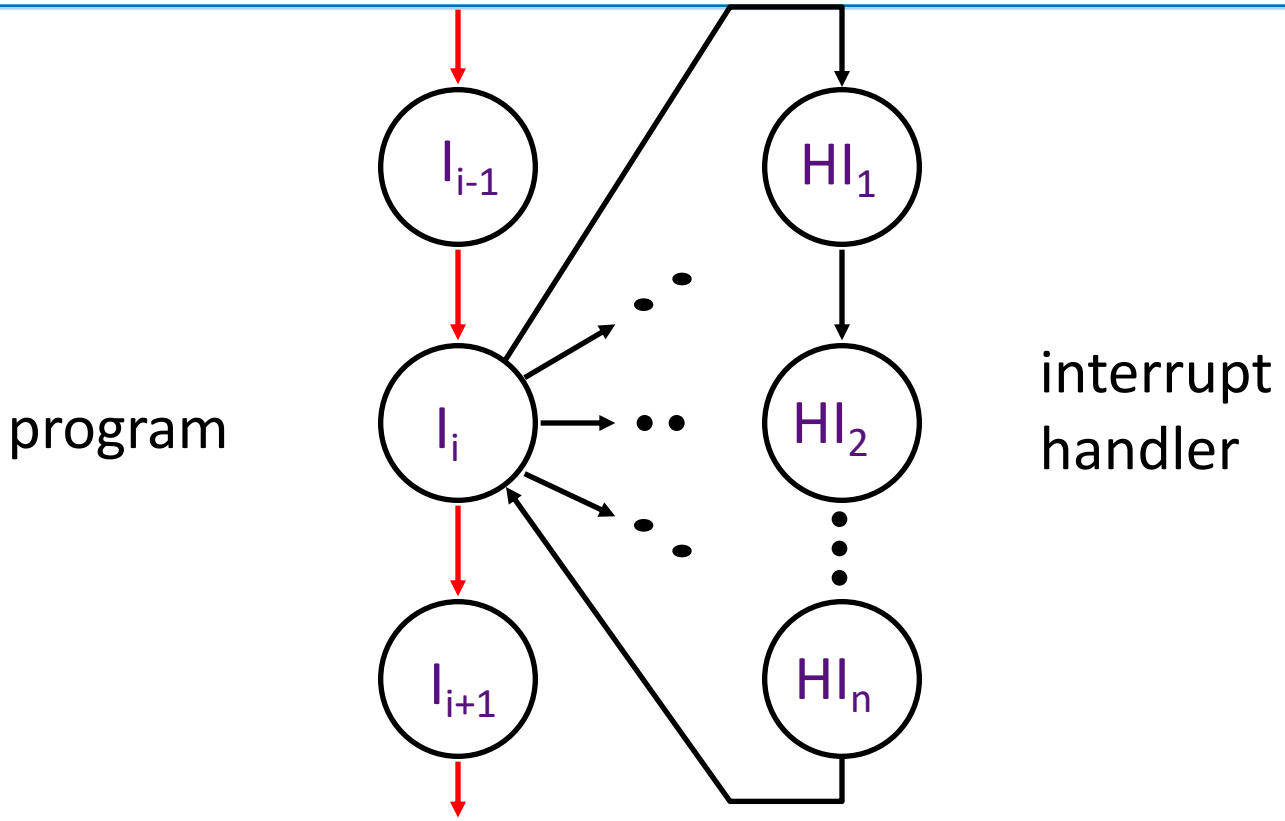
Calculatoare Numerice

– Cursul 13 –

Întreruperi

Facultatea de Automatică și Calculatoare
Universitatea Politehnica București

Întreruperi: schimbarea fluxului normal de execuție



O **întrerupere** este un eveniment *extern* sau *intern* care trebuie procesat de o rutină (sau program) separată. Evenimentul este de obicei neașteptat (asincron față de ceasul sistemului) și/sau rar din punctul de vedere al programului principal.

Surse pentru întreruperi

Embedded Systems Laboratory

Întrerupere: un *eveniment* ce necesită atenția procesorului

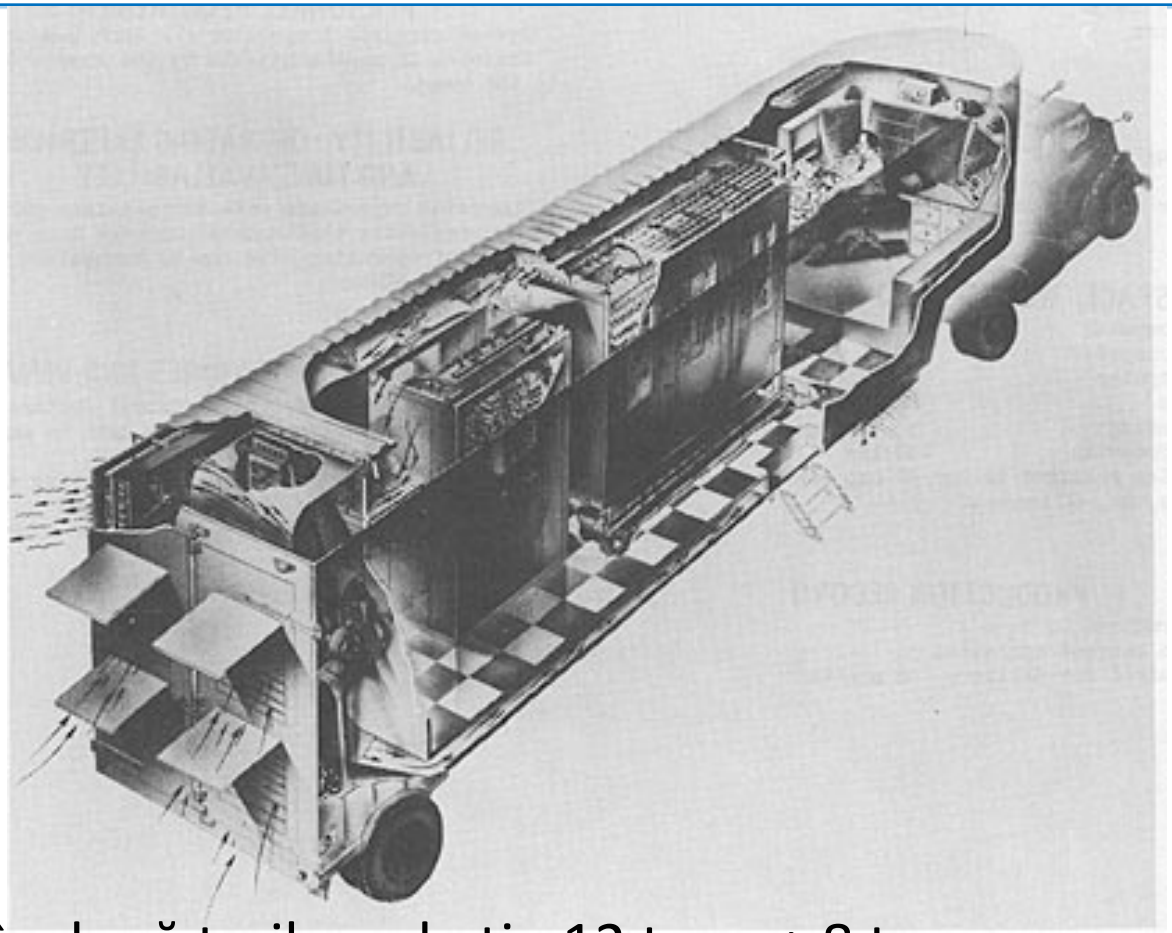
- Asincron: un eveniment extern
  - Cerere de la un dispozitiv input/output
  - Expirarea unui timer
  - Întrerupere în alimentarea cu energie, cădere hardware
- Sincron: un eveniment intern (*a.k.a. Trap sau Excepție)*
  - Opcode nedefinit, instrucțiune privilegiată
  - Overflow aritmetic, excepție FPU
  - Acces prost aliniat la memorie
  - *Excepții de memorie virtuală:* page faults, TLB misses, protection violations
  - System calls (e.g., jump-uri în kernel)

O scurtă istorie pentru exception handling

Embedded Systems Laboratory

- Primul sistem care trata excepțiile Univac-I, 1951
  - Overflow aritmetic, care avea următoarele roluri:
    1. Declanșează execuția unei rutine de fix-up de două instrucțiuni situată la adresa 0, sau
    2. La alegerea programatorului, cauzează mașina de calcul să se oprească
  - Mai târziu, Univac 1103, 1955, modificat pentru a trata întreruperi externe
    - Folosit pentru a culege informații de timp real dintr-un tunel aerodinamic
- Primul sistem cu întreruperi pe I/O a fost DYSEAC, 1954
  - Avea două PC-uri și o întrerupere de la I/O cauza schimbarea PC-ului curent
  - De asemenea, era primul sistem cu DMA (Direct Memory Access pentru un dispozitiv I/O)

DYSEAC, primul computer mobil!



- Instalat în două trailere de tir, 12 tone + 8 tone
- Construit pentru US Army Signal Corps *[Courtesy Mark Smotherman]*

Întreruperi asincrone: invocarea rutinei de tratare a întreruperii

Embedded Systems Laboratory

- Un dispozitiv I/O cere atenție prin semnalizarea pe o linie dedicată de interrupt request (IRQ)

- Când un procesor decide să proceseze întreruperea
  - Oprește execuția programului curent la instrucțiunea I_i, completând execuția tuturor instrucțiunilor până la I_{i-1} (*întrerupere precisă)*
  - Salvează PC-ul instrucțiunii I_i într-un registru special (EPC)
  - Maschează toate întreruperile și transferă controlul unei rutine speciale pentru tratarea întreruperii (pentru un OS ca Linux/Windows aceasta e situată de obicei în kernel)

Rutina de tratare a întreruperii

- Salvează EPC înainte să permită execuția altor întreruperi (întreruperi imbricate) $\Rightarrow$
  - Avem nevoie de o instrucțiune pentru a muta EPC în GPR-uri
  - Avem nevoie de un mecanism pentru a masca toate întreruperile până când EPC este salvat
- Trebuie să citească un Registru de Stare care indică ce periferic sau subsistem a cauzat întreruperea
- Folosește o instrucțiune specială de salt indirect RFE (*return-from-exception*) care
  - Demaschează întreruperile
  - Întoarce procesorul înapoi în modul utilizator
  - Reface statusul inițial hardware și starea unității de control

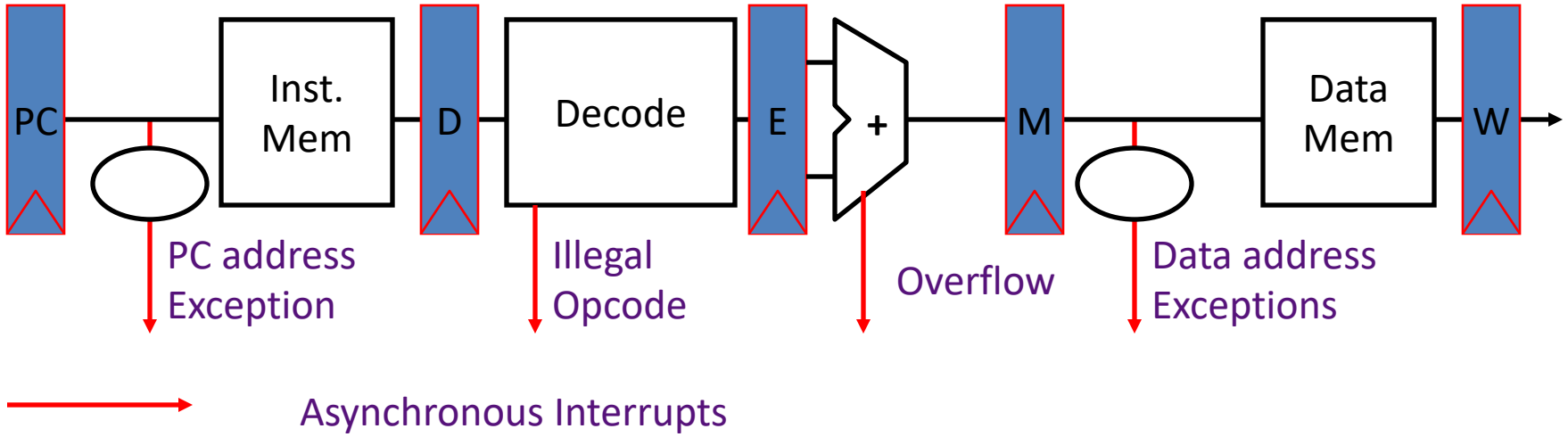
Întreruperi sincrone

- O întrerupere sincronă (excepție) este cauzată de o anumită instrucțiune

- În general, instrucțiunea nu poate fi completată și trebuie să fie restartată după tratarea excepției pe care a generat-o
  - Necesită refacerea efectului unei sau mai multor instrucțiuni executate parțial

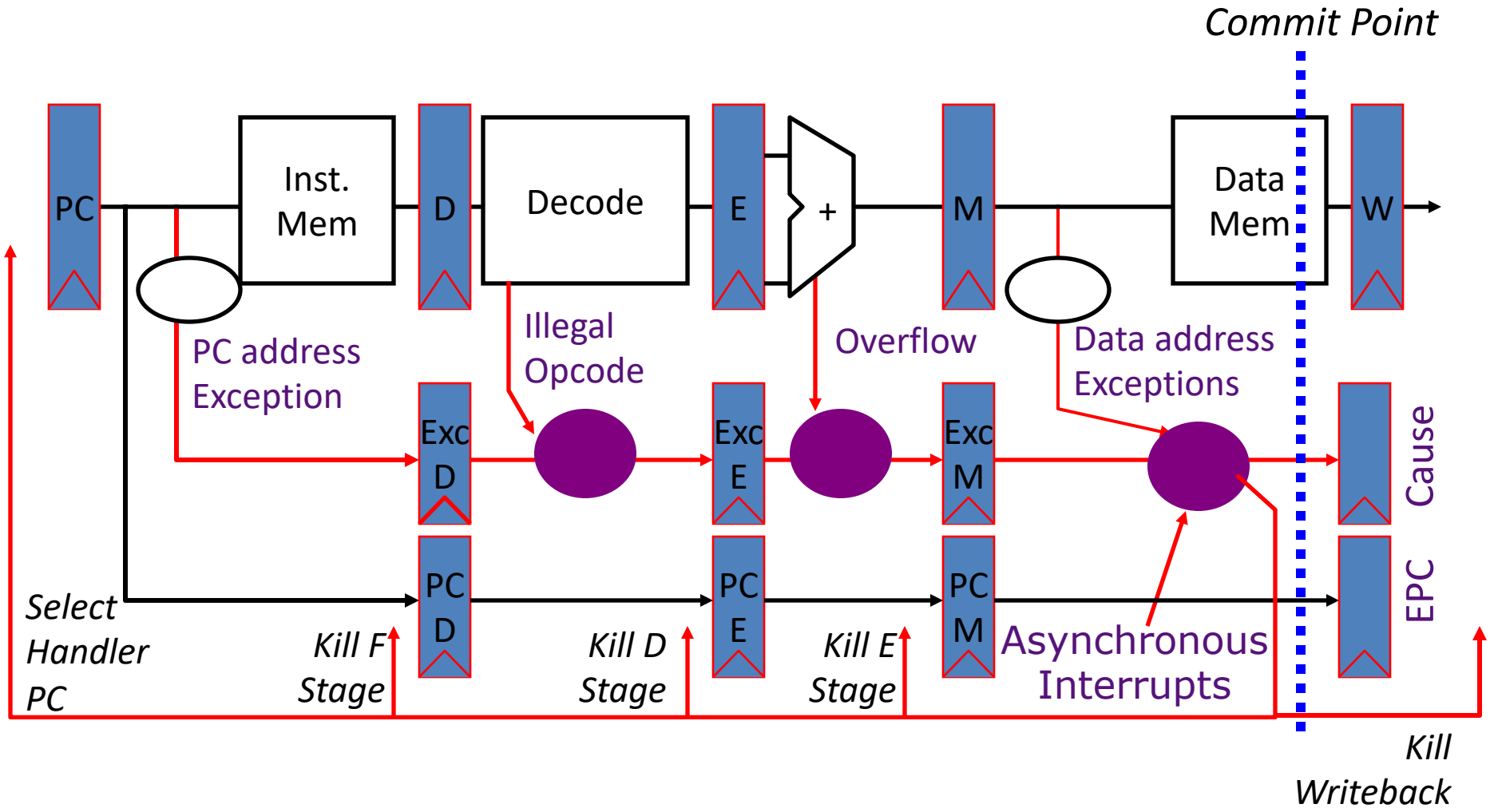
- În cazul unui system call trap, se consideră că instrucțiunea și-a incheiat execuția
  - O instrucțiune de jump specială ce face o trecere a procesorului în modul kernel privilegiat

Tratarea întreruperilor b.a. cu 5 etape



- Cum putem să tratăm excepții multiple simultane în diferitele etape ale b.a.?
- Cum și unde putem să tratăm întreruperile asincrone externe?

Tratarea întreruperilor b.a. cu 5 etape



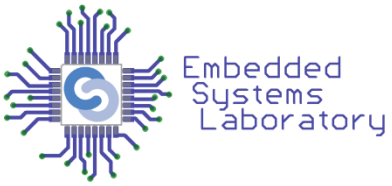
Tratarea întreruperilor b.a. cu 5 etape

- Ținem flag-urile de excepție în b.a. până la commit point (etapa M)
- Excepțiile din etapele anterioare suprascriu excepțiile din etapele superioare pentru o instrucțiune dată
- Injectează întreruperile externe la commit point (suprascrie toate celalalte)
- Dacă excepția se petrece la commit: actualizează registrele Cause și EPC, invalidează toate etapele din b.a., injectează adresa rutinei de tratare a întreruperii în PC

Speculații la excepții

- Mecanism de predicție
  - Excepțiile sunt foarte rare, așa că un mecanism care prezice că nu există nici o excepție este foarte precis!
- Verificarea mecanismului de predicție
  - Excepțiile sunt detectate la sfârșitul execuției în b.a., hardware special pentru diferitele tipuri de excepții
- Mecanism de revenire din excepție
  - Scriem doar starea generală la commit point, ca să putem să neglijăm toate instrucțiunile executate parțial după apariția excepției
  - Lansează rutina de tratare a întreruperii după ce golim întreaga bandă de asamblare

Diagrame b.a. pentru tratarea excepțiilor



	timp t0	t1	t2	t3	t4	t5	t6	t7	. . . .
(I_1) 096: ADD	IF_1	ID_1	EX_1	MA_1	-		*overflow!*		
(I_2) 100: XOR		IF_2	ID_2	EX_2	-	-			
(I_3) 104: SUB			IF_3	ID_3	-	-	-		
(I_4) 108: ADD				IF_4	-	-	-	-	
(I_5) Exc. Handler code					IF_5	ID_5	EX_5	MA_5	WB_5

Resource Usage

	timp t0	t1	t2	t3	t4	t5	t6	t7	. . . .
IF	I_1	I_2	I_3	I_4	I_5				
ID		I_1	I_2	I_3	-	I_5			
EX			I_1	I_2	-	-	I_5		
MA				I_1	-	-	-	I_5	
WB					-	-	-	-	I_5

Apeluri de sistem pentru I/O

Embedded Systems Laboratory

De ce utilizatorul trebuie să fie izolat de toate detaliile operațiilor I/O?

Protecție: Utilizatorul trebuie să nu acceseze anumite zone de memorie

Convenienţă: Nu trebuie să știe toate detaliile de operare pentru fiecare device

Eficienţă: Majoritatea utilizatorilor sunt incapabili să găsească cel mai bun mod de acces la I/O

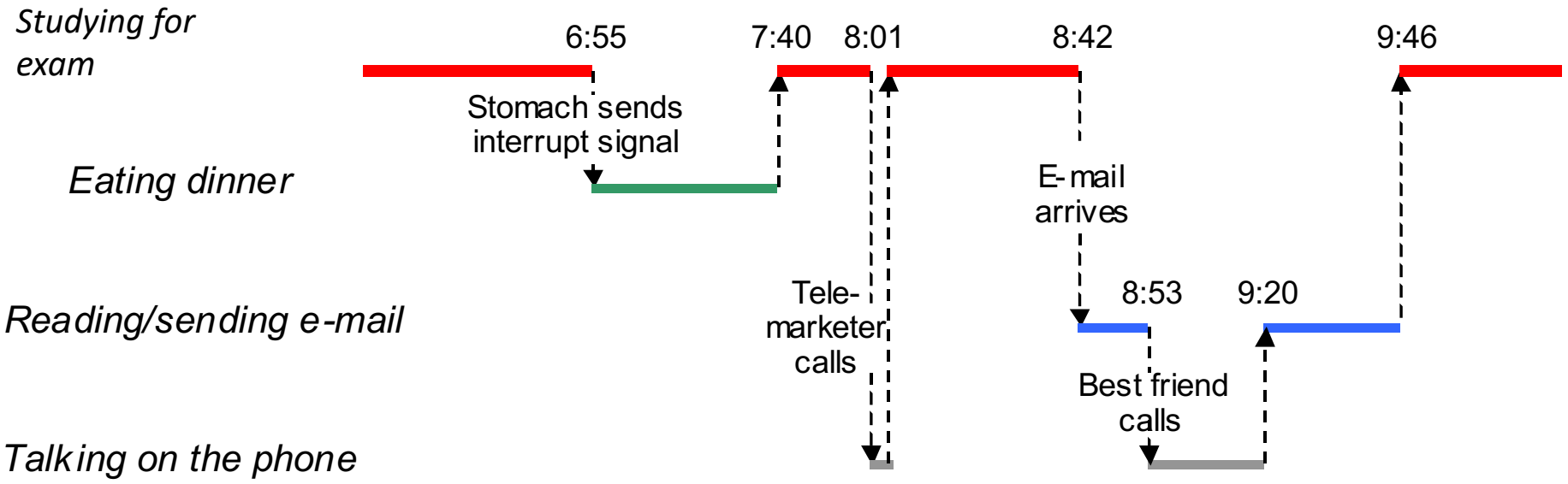
Abstractizarea I/O: grupăm dispozitivele I/O într-un număr mic de clase generice pentru a face operațiile de I/O independente de dispozitive

Character stream I/O: get(•), put(•) – e.g., keyboard, printer
Block I/O: seek(•), read(•), write(•) – e.g., disk

Network Sockets: create socket, connect, send/receive packet
Clocks or timers: set up timer (notificare prin întrerupere)

Întreruperi, Excepții și System traps

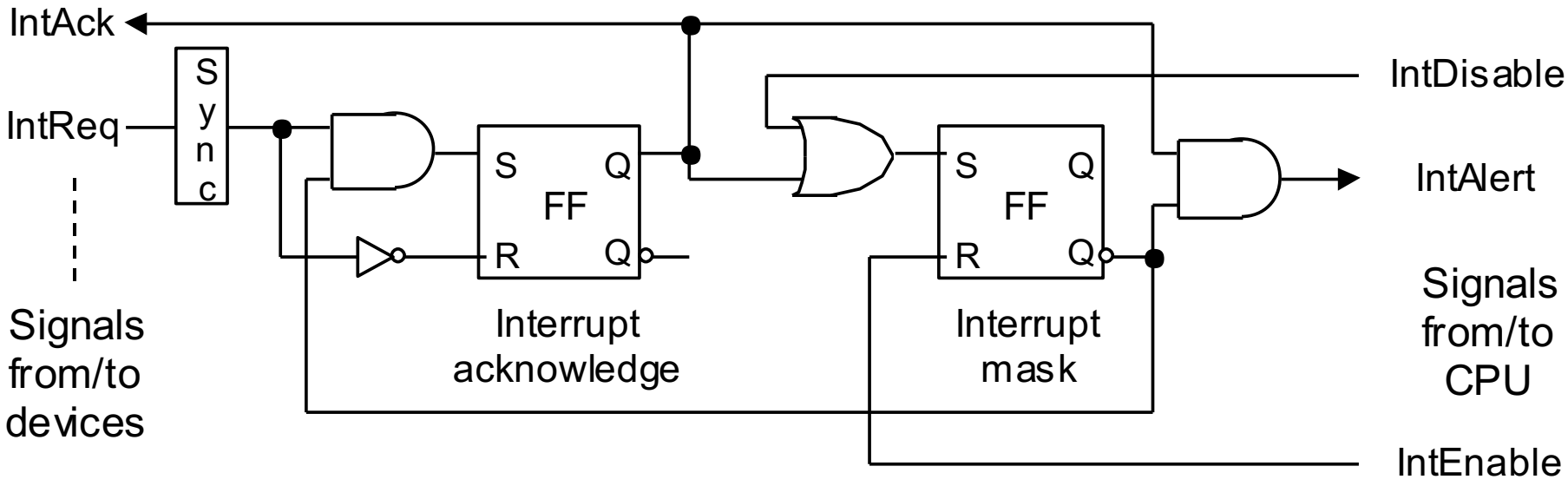
Intrerupere	Termen general pentru orice cerere sau de la I/O
Excepție	Cauzate de o operație ilegală (imprevizibile)
Trap	AKA “software interrupt” (pre-planificată)



Noțiunea de întrerupere și întreruperi imbricate

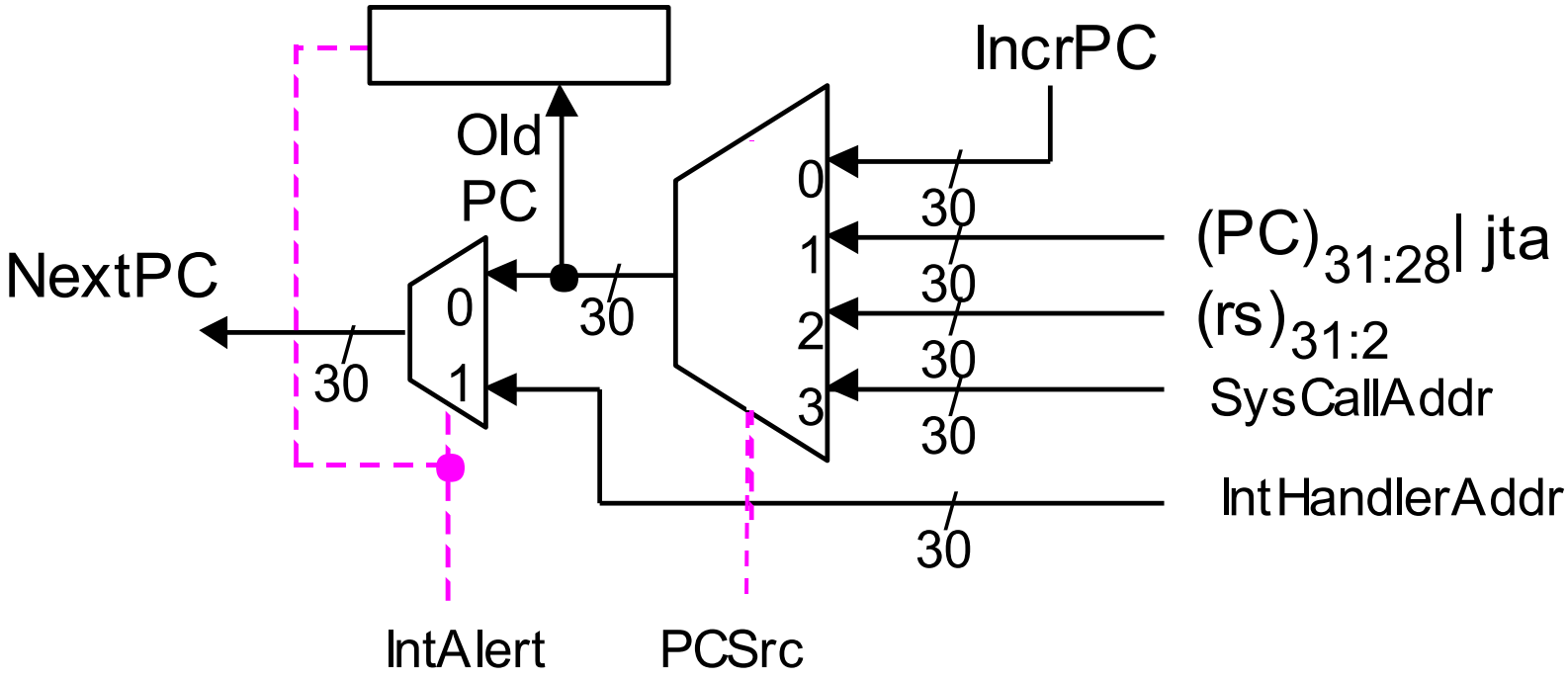
Tratarea întreruperilor

Răspundem la cererea de întrerupere prin semnalul IntAck
Notificăm procesorul că avem o întrerupere de tratat
Setăm masca de întreruperi a.î. noua întrrupere să fie demascată



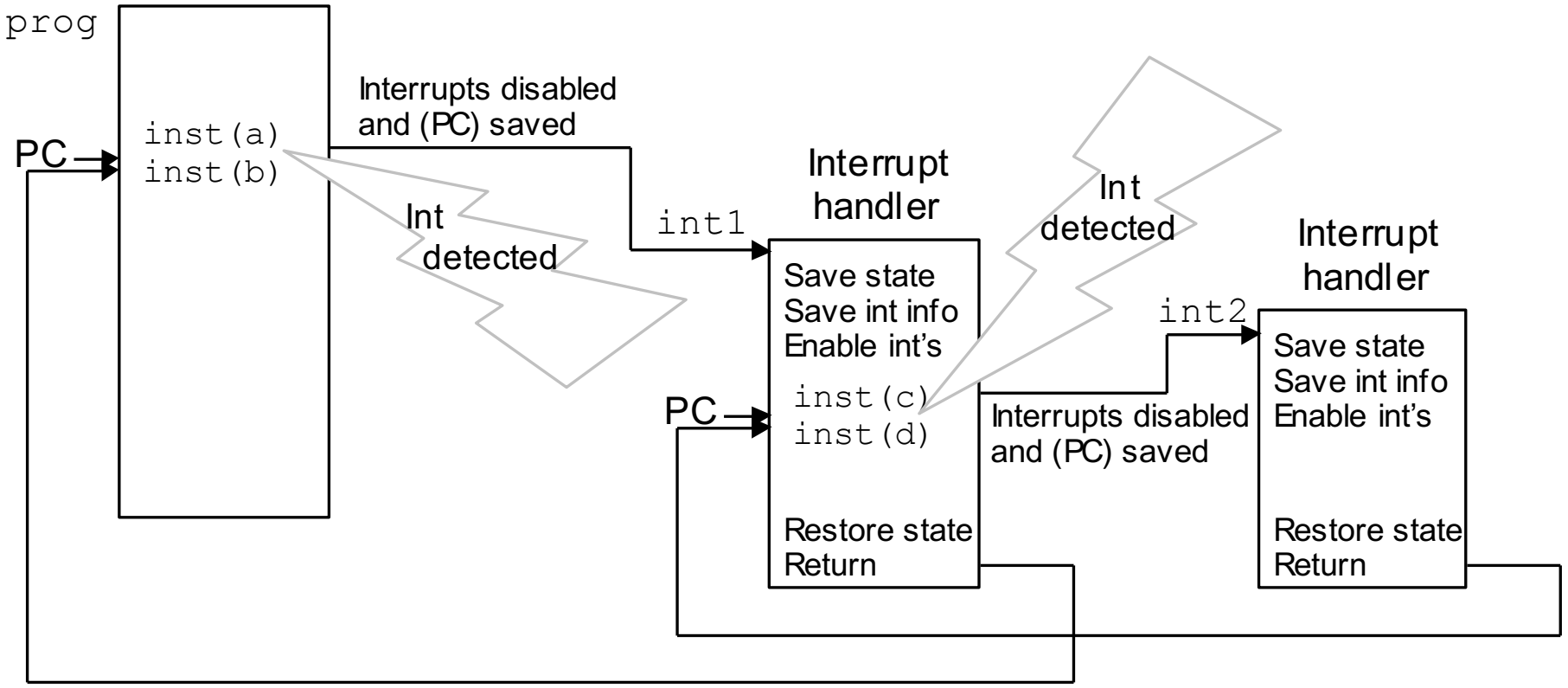
Logică simplă de semnalizarea a unei întreruperi pentru un procesor

Logica pentru calculul rutinelor de tratare a întreruperilor



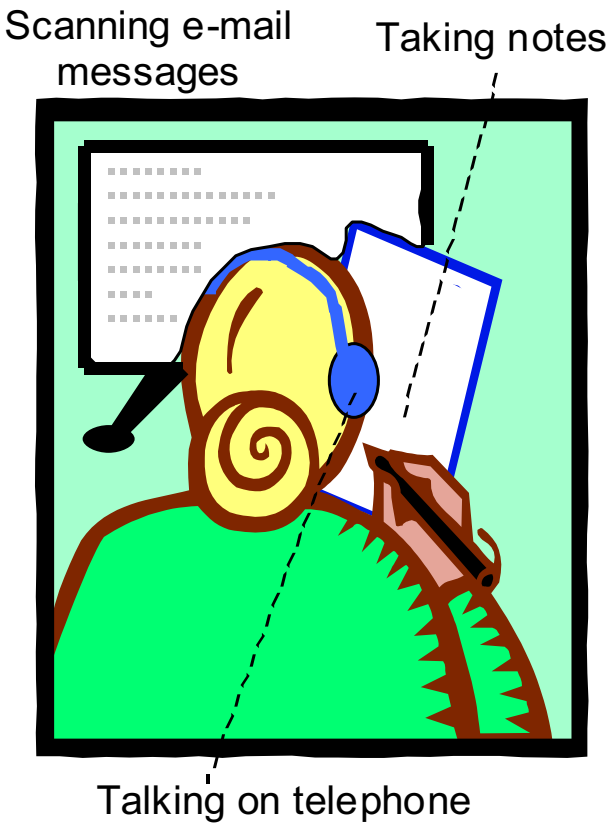
Logica pentru calculul adresei următoare de program la care s-a adăugat calculul adreselor pentru rutinele de tratare a întreruperilor

Întreruperi imbricate

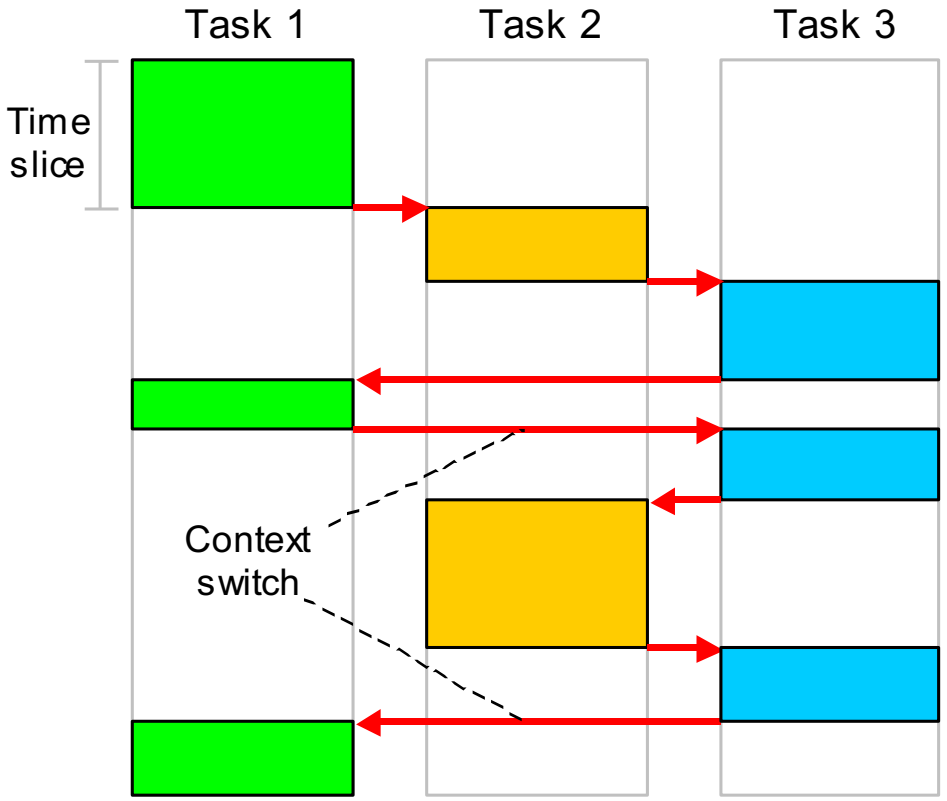


Exemplu de întrerupere imbricată.

Context switching



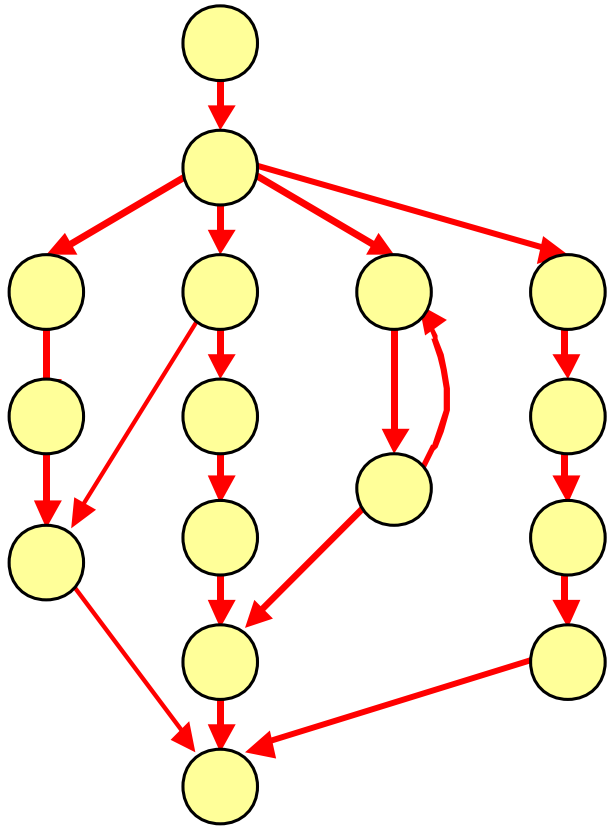
(a) Human multitasking



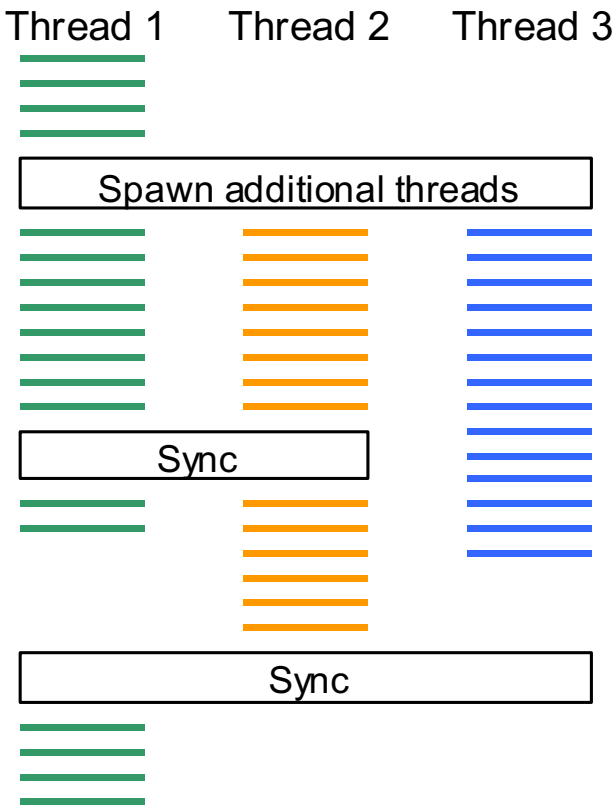
(b) Computer multitasking

Multitasking la oameni și la calculatoare

Threaduri și multithreading



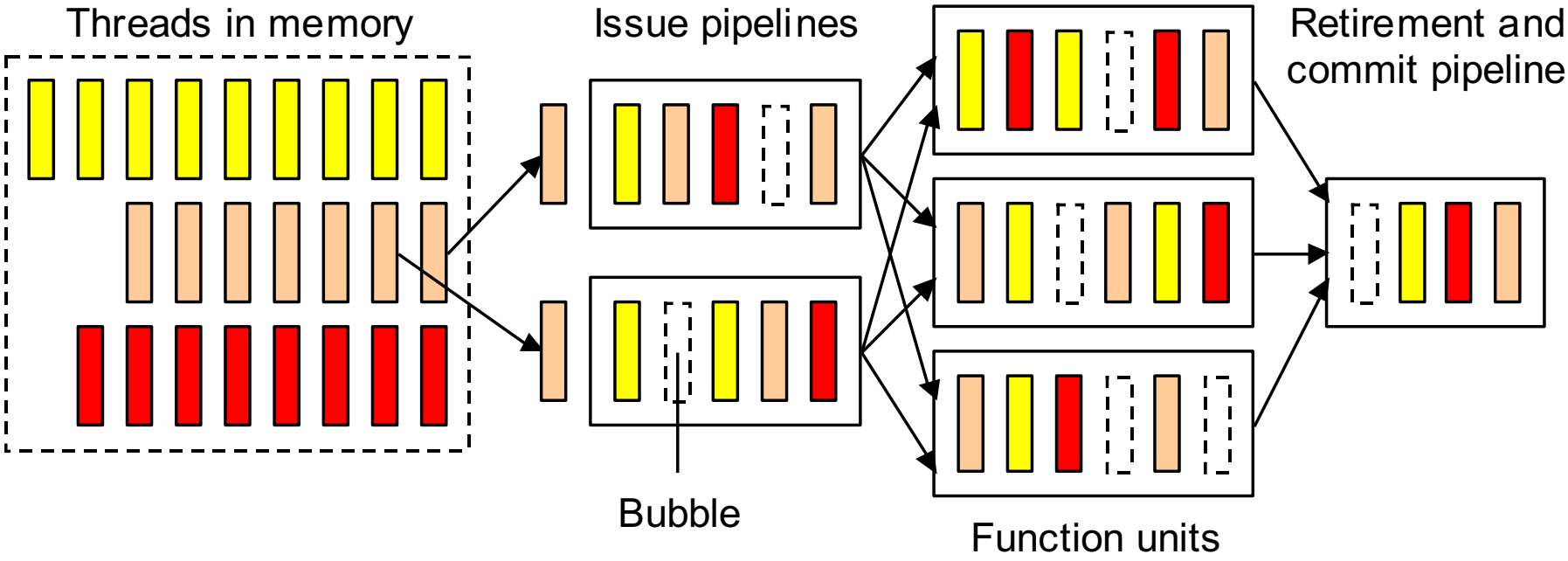
(a) Task graph of a program



(b) Thread structure of a task

Un program poate fi împărțit în task-uri sau thread-uri

Procesoare cu mai multe fire de execuție



Instrucțiuni din mai multe thread-uri pe măsură ce își găsesc drumul prin banda/benzile de asamblare ale unui procesor modern.

Unde mergem mai departe?

Proiectarea memoriei:
Cum putem proiecta o memorie care să răspundă într-un singur ciclu de ceas?

Input and Output:
Periferice, programarea I/O, interfațare, întreruperi

Performanță (și) mai mare:
Procesare vectorială
Procesare paralelă