Apunts del Tema 1
Introducció

Joan Manuel Parcerisa

Departament d'Arquitectura de Computadors
Facultat d'Informàtica de Barcelona
Març 2020

Licencia Creative Commons

Tema 1. Introducció

1.2, 1.3

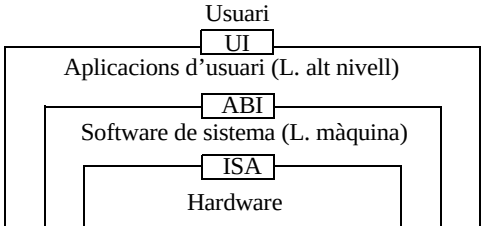
1. Descripció jeràrquica del computador

El hardware d'un computador realitza operacions molt simples i de baix nivell. El disseny de hardware o software emprant directament aquestes operacions seria d'una complexitat intractable. Per a fer front a aquesta complexitat, els dissenyadors de hardware i software utilitzen el mecanisme d'*abstracció*[1] per descompondre un sistema en capes disposades de forma jeràrquica, situant-se a la base de la jerarquia les operacions reals de més baix nivell, i oferint cada nova capa un model de descripció que simplifica els detalls de la capa anterior:

- Hardware: Portes lògiques, circuits combinacionals/seqüencials. Adequat per a un dissenyador de computadors (IC).
- Llenguatge màquina (MIPS, x86, ARM, PowerPC): instruccions, registres, modes d'adreçament, model de memòria, etc. Adequat per a un programador de sistemes (EC).
- Llenguatge d'alt nivell (C, C++, Fortran, Java, Prolog, LISP, RPG, etc.): variables, sentències, funcions, etc. Adequat per a un programador de software (Pro2).
- Llenguatge d'usuari: comandes, menús, icones, clic-esqerre, arrastrar, seleccionar, etc. Adequat per a un usuari final (curs d'ofimàtica)

1.1 Llenguatges i interfícies. Cas particular del ISA/ABI

Cada nivell estableix una interfície amb la qual el nivell superior pot interactuar amb el computador sense conèixer cap més detall de la implementació (caixa negra):



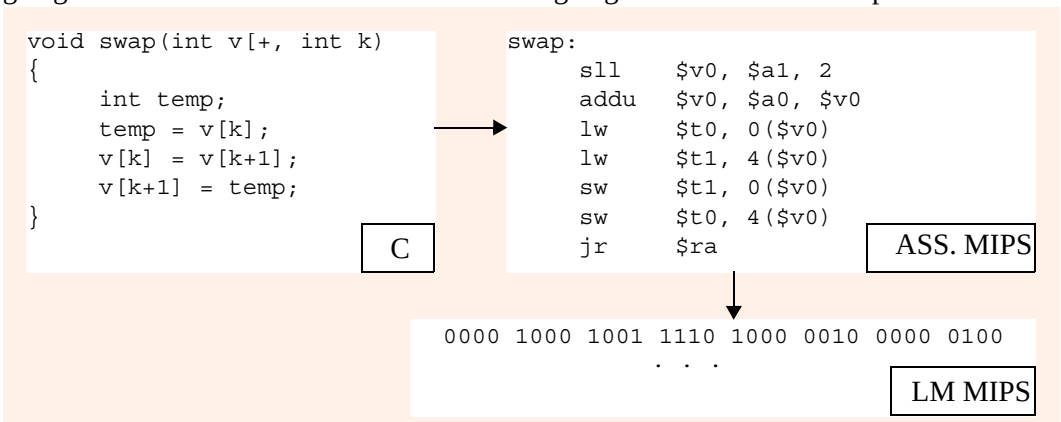
1. Abstracció: Procés consistent en, partint de les característiques comunes que poden manifestar-se en un sistema, separar-les de les altres, menys rellevants, establint un nou model, més simple (exemple: descripció d'un automòvil).

a) ISA. El nivell hardware estableix una interfície anomenada *Instruction Set Architecture* (ISA). Es tracta d'una especificació que descriu els aspectes del processador visibles a un programador de llenguatge màquina (o assemblador): el repertori d'instruccions, amb la seva sintaxi i format, els registres, el model de memòria, entrada/sortida, excepcions, així com el comportament del sistema quan s'executen cada una de les possibles instruccions. La ISA abstrau els detalls de la implementació (microarquitectura), establint un compromís de funcionament que ha de complir qualsevol implementació hardware per a poder ser programada en el llenguatge que es descriu. Gràcies a això, una mateixa ISA pot ser implementada de diferents maneres, però els programes es podran executar en totes. Per exemple, l'Intel Core i l'AMD Athlon implementen versions gairebé idèntiques del conjunt d'instruccions x86, encara que tenen dissenys interns diferents.

b) ABI. El nivell de Software del Sistema (escrit en llenguatge màquina) estableix una interfície anomenada *Application Binary Interface* **(**ABI**)**. Es tracta d'una especificació que descriu les funcions del Sistema Operatiu, tals com la gestió de la memòria, l'emmagatzematge de programes i dades o la compartició segura dels recursos del computador entre múltiples programes concurrents. Algunes d'aquestes funcions es troben en el nucli, i d'altres en biblioteques de funcions (*libraries*). La interfície del SO descriu l'accés als seus serveis per mitjà de les crides al sistema, i estableix els convenis de crida i retorn de funcions, pas de paràmetres, etc. que han de complir els programes per interactuar amb les funcions de biblioteca del sistema. De fet, en la majoria de casos, l'encarregat d'assegurar que els programes compleixin les regles d'interfície establertes per l'ABI és el compilador, ja que aquest és qui tradueix un programa escrit en llenguatge d'alt nivell, a llenguatge màquina fent correctament les crides a serveis i/o biblioteques del sistema. Per uniformitat, el compilador aplica les mateixes regles a les crides entre funcions de una mateixa aplicació. L'ABI abstrau els detalls de la implementació del SO, establint un compromís de funcionament que ha de complir qualsevol implementació del sistema per a interactuar correctament amb els programes. Gràcies a això, mentre el fabricant del sistema operatiu respecti íntegrament l'especificació de l'ABI, pot modificar-ne la implementació sense afectar a la garantia de bon funcionament dels programes existents.

1.2 Traducció: Compilació vs Interpretació

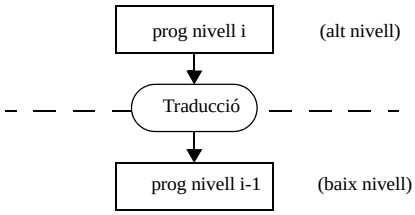
Les accions (instruccions, sentències, comandes o clics) que s'executen en el llenguatge de nivell *i* han de ser convertides al llenguatge del nivell *i*-1. Exemple:



Nota: el llenguatge assemblador i màquina són del mateix nivell, es genera 1 instrucció de LM per cada línia assemblador. La diferència és que l'assemblador és fàcilment

llegible pels humans (les operacions són *mnemònics* i els operands números decimals en comptes de tires de bits, i estan clarament separats per comes, parèntesis o espais).

La traducció es pot fer de 2 maneres: per compilació o per interpretació:



a) Compilació: Un programa traductor que converteix tot el *prog-i* en un *prog-i*-1. Però no l'executa, sols l'emmagatzema. Exemple: Compilador de C a assemblador MIPS.

- Pros: La compilació admet moltes optimitzacions i pot assolir el màxim rendiment possible del programa. A pesar que la traducció és molt costosa, sols s'ha de traduir 1 cop (mentre no s'alteri el codi font).
- Contras: El programa s'ha de recompilar cada cop que es vol adaptar a un sistema (ISA i/o ABI) diferent. Escriure un compilador és molt complex,

b) Interpretació: Un programa intèrpret llegeix una per una les accions de *prog-i* i executa una seqüència equivalent d'accions en el llenguatge *i*-1. No emmagatzema res, sols l'executa. Exemples: Shell Unix, Intèrpret Python, MV Java, Unitat de control de la CPU (interpreta instruccions de codi màquina), etc.

- Pros: Escriure un intèrpret és relativament senzill. Un mateix programa es pot portar d'un sistema a un altre diferent sense necessitat d'adaptar-lo (sols cal que els dos sistemes tinguin l'intèrpret corresponent instal·lat).
- Contras: El *prog-i* s'ha de traduir cada cop que es vol executar: és més lent

Resumint idees importants:

- Escriure programes en alt nivell pot produir codi menys *eficient*, però aporta més *productivitat*, *fiabilitat* i *portabilitat*.
- La interpretació facilita que programes i plataformes *evolucionin* independentment, adaptant-se mútuament, però l'execució és massa lenta per a aplicacions on la velocitat és crítica, i llavors és millor compilar-les.

EXEMPLE 1**:** Tenim un computador MIPS ¿Com podem executar-hi un programa escrit en Java si disposem del següent software?

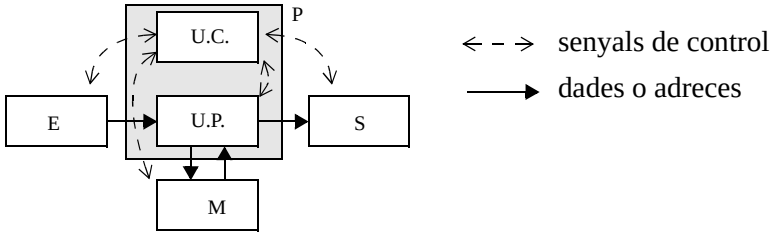
- Compilador de C a x86 (escrit en x86)
- Traductor binari de x86 a MIPS (escrit en MIPS)
- Intèrpret de Java (escrit en C)

Solució:

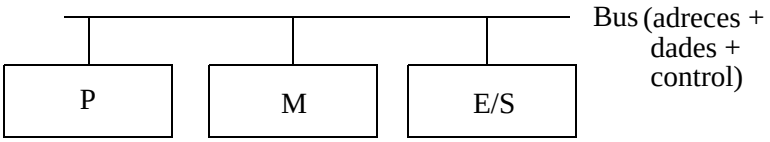
- Traduir el binari del compilador de C, de x86 a MIPS (compila de C a x86)
- Compilar l'intèrpret de Java, de C a x86
- Traduir el binari de l'intèrpret de Java, de x86 a MIPS
- Executar l'intèrpret de Java i subministrar-li el programa Java com entrada.

1.3 El model de computador Von Neumann

Von Neumann, enginyer del projecte ENIAC, un dels primers computadors construïts, va escriure el 1945 l'informe[2] on es descrivia una proposta per al disseny del nou projecte EDVAC, amb idees compartides per A.Turing, J.P.Eckert i J.Mauchley (les idees van ser posteriorment realitzades en molts altres computadors però l'EDVAC no es va construir realment fins molts anys més tard). El seu model està caracteritzat per emmagatzemar les instruccions del programa en el mateix espai d'adreces de memòria on es guarden les dades. El model es compon de 5 parts, que ell va anomenar input (I), output (O), memory (M), central arithmetic (CA, avui coneguda com Unitat de Procés) i central control (CC, avui coneguda com Unitat de Control). La unitat de procés conjuntament amb la unitat de control conjuntament formen el Processador (o CPU):



Sovint, el model es presenta de forma simplificada, amb sols tres blocs: Processador, Memòria i dispositius d'Entrada/Sortida, connectats per un Bus amb tots els senyals:



1.4

2. Mesura del Rendiment

2.1 Definició de "Rendiment"

El disseny de computadors aspira a millorar-ne contínuament el rendiment. Però ens cal definir amb precisió aquest concepte a fi de ser capaços de mesurar si els nous dissenys aporten millores o no.

a) Temps d'execució o productivitat

El terme "rendiment" és usat per a referir-se a dos conceptes diferents:

- Temps d'execució (o de resposta): temps transcorregut entre l'inici i el final d'una tasca. És el temps que experimenta l'usuari. El rendiment és l'invers d'aquest temps.
- Productivitat (throughput): Nombre de tasques completades per unitat de temps. Usat sovint per quantificar el rendiment en centres de dades.

2. John von Neumann. "First Draft of a Report on the EDVAC". University of Pennsylvania, June 30, 1945

EXEMPLE 2: un avió A porta 100 passatgers a 1200 Km/h. Un avió B porta 500 passatgers a 1000 Km/h. Quin té més rendiment? Resposta: A té menor temps de resposta. B té major productivitat

EXEMPLE 3: un computador A té 1 CPU que tarda 10 us. B té 200 CPUs que tarden cada una 20 us. Quin té més rendiment? Resposta: A té menor temps d'execució. B té major productivitat.

Quina relació hi ha entre productivitat i temps d'execució?

- A menor temps d'execució, més productivitat (obvi)
- Augmentar la productivitat pot disminuir el temps d'execució, però només en cas de congestió, ja que es redueix el temps que romanen les tasques encuades (com en el cas d'un datacenter)

b) Temps de CPU

El temps d'execució (o de resposta) d'un programa consisteix en la suma de diverses components: Temps de CPU + Temps d'espera d'E/S + Temps encuat mentre s'executen altres programes concurrents. En endavant, en aquest curs d'EC, considerarem com a temps d'execució exclusivament el *temps de CPU*, ja que és el que depèn més directament del disseny de la CPU.

Resumint: Definicions de *Rendiment* i de *Guany de rendiment* (o *speedup*)

Així doncs definim el Rendiment com la inversa del temps d'execució:

$$\text{Rendiment} = 1 / t_{exe}$$

El Guany de rendiment (speedup) o rendiment relatiu ens diu quantes vegades més ràpid s'executa un programa després d'introduir una millora (en el programa o en l'arquitectura). El guany (denotat per *s*) del cas millorat respecte de l'original és el quocient entre els rendiments respectius, és a dir el quocient dels inversos dels temps d'execució:

$$\mathbf{s} = \text{Rendiment}_{\text{millorat}} / \text{Rendiment}_{\text{original}} = \mathbf{t_{original} / t_{millorat}}$$

2.2 Factors que influeixen en el temps d'execució

Per tal d'entendre quins aspectes influeixen en el temps d'execució, i per tant conèixer com és possible millorar-lo, el desglossarem en els seus components.

$$t_{exe} = n_{cicles} \cdot t_c = n_{cicles} / f_{clock}$$

On t_{exe} és el temps d'execució (de CPU), n_{cicles} el número de cicles de rellotge que tarda l'execució i t_c i f_{clock} són el període i la freqüència del rellotge. El rendiment depèn del número de cicles i de la freqüència. Considerem en primer lloc el número de cicles.

a) Reduir n_{cicles}

Intuitivament, veiem dos camins per a reduir-lo: en primer lloc, fer que el programa s'executi amb menys instruccions (n_{ins}), i en segon lloc, fer que aquestes tardin el mínim nombre de cicles per instrucció (CPI). És a dir, essent CPI el promig de cicles per instrucció de tot el programa, el número de cicles total es pot expressar com el producte

$$n_{cicles} = n_{ins} \cdot CPI$$

I per tant, el temps d'execució és

$$t_{exe} = n_{ins} \cdot CPI \cdot t_c$$

Per a reduir n_{ins} cal millorar l'eficiència del compilador. El CPI d'un programa determinat és un promig que depèn del retard de cada tipus *i* d'instrucció (CPI_i) i de quantes n'hi ha de cada tipus (n_i), fent una mitjana ponderada. Així, suposant que té *m* tipus diferents d'instruccions, el seu CPI és:

$$CPI = \frac{1}{n_{ins}} \cdot \sum_{i=1}^{m} n_i \cdot CPI_i$$

I per tant, el temps d'execució és

$$t_{exe} = \left(\sum_{i=1}^{m} n_i \cdot CPI_i \right) \cdot t_c$$

Podem reduir el temps d'execució reduint el retard CPI_i de cada tipus d'instrucció, però això requereix millorar el disseny de la microarquitectura. També podem reduir el número d'instruccions n_i d'aquells tipus d'instruccions més lentes (amb major CPI_i) encara que sigui a costa d'augmentar el nombre de les més ràpides, si la suma és inferior, millorant l'habilitat del compilador.

Cal anar amb compte amb el concepte CPI, perquè pot significar:

- CPI *promig d'un programa o conjunt de programes* en un cert computador.
- CPI *fix d'un tipus d'instruccions* en un cert computador (CPI_i per al tipus *i*): Donat un determinat computador, cada tipus d'instrucció tarda un cert nombre de cicles, i en molts casos és un nombre fix per a cada tipus[3].

Comparant el rendiment de dues versions d'un mateix programa generades per compiladors diferents estem comparant l'efectivitat dels compiladors. Com que el temps de cicle és el mateix, la comparació dels temps d'execució es pot fer comparant el nombre de cicles de rellotge.

EXEMPLE 4: Volem comparar dues versions d'un programa en un mateix computador, que disposa de 3 tipus d'instruccions A, B i C, amb CPI de 1, 2, 3 respectivament. El programa 1 consta de: 2 instruccions d'A, 1 de B i 2 de C. El programa 2 consta de: 4 instruccions d'A, 1 de B i 1 de C. Quin és més ràpid? Quins són els CPI?

Solució: Calculem el núm d'instruccions i cicles totals de cada programa omplint les dues columnes a la dreta d'aquesta taula:

tipus	**CPI**	**nins**		**ncicles**	
		p1	**p2**	**p1**	**p2**
A	1	2	4	2	4
B	2	1	1	2	2
C	3	2	1	6	3
Total			6	10	9

El més ràpid és p2 (9 cicles en lloc de 10).

3. Però alerta, perquè no sempre és cert que un tipus d'instrucció sempre tardi un temps fix, ja que com veurem al tema 6, els loads i stores d'un mateix programa poden tardar més o menys depenent de l'estat de la cache en cada instant. No obstant, fins i tot en aquest cas, encara podem parlar del CPI dels loads, referint-nos llavors al promig dels loads dins d'un determinat programa.

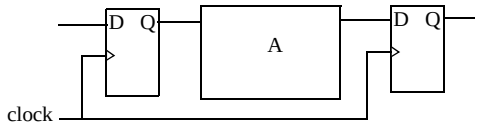
Calculem ara els CPI

CPI_1 = 10cicles / 5instruccions = 2
CPI_2 = 9cicles / 6instruccions = 1,5

Observem que p2 té més instruccions però menys cicles per instrucció

b) Augmentar la freqüència de rellotge (reduir el temps de cicle)

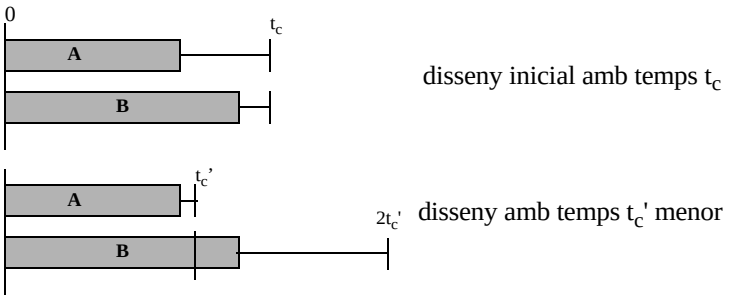
Suposem un circuit combinacional A que llegeix dades d'entrada d'un biestable i escriu dades de sortida en un altre biestable.



En principi, reduint el temps de cicle del circuit A podem aconseguir que calculi més resultats per unitat de temps, reduint doncs el temps total d'execució de les tasques.

Però cal parar atenció: si reduïm excessivament el temps de cicle en l'anterior circuit, pot ser que les dades de sortida de A encara no estiguin calculades, i que el resultat correcte no s'obtingui fins al següent cicle de rellotge. En aquest cas, haurà augmentat el nombre de cicles per executar la mateixa tasca, i no és segur que hi hàgim guanyat.

Per exemple, suposem dos circuits A i B de diferent retard, que formen part del mateix processador i escriuen repetidament el seu resultat sobre un biestable quan arriba el flanc de rellotge. Suposem que inicialment el temps de cicle és t_c i volem reduir-lo a t_c':



Després de reduir el temps de cicle, la tasca A es realitzarà en menys temps, ja que no canvia el número de cicles, però la tasca B no podrà escriure en el primer cicle i necessitarà un cicle més: necessitarà el doble de cicles que abans. La reducció de t_c només suposarà una millora de rendiment si la influència del cas A en el temps d'execució total és major que la de B. Veiem-ho amb un parell d'exemples:

EXEMPLE 5: El processador A té tc_A = 500 ps i CPI_A = 2. Suposem que el redissenyem per tal que usi un temps de cicle menor: el nou processador B té t_{cB} = 250 ps. Però malauradament això comporta augmentar el nombre de cicles del programa de test, és a dir que augmenta el CPI promig: CPI_B = 3. Serà més ràpid el nou disseny?

Solució: Siguin t_A, t_B els temps d'execució i n el nombre de cicles del programa, llavors el guany (speedup) de B respecte de A és:

$$s_B = t_A / t_B = (n \cdot CPI_A \cdot t_{cA}) / (n \cdot CPI_B \cdot t_{cB})$$
$$= (CPI_A \cdot t_{cA}) / (CPI_B \cdot t_{cB}) = (2 \cdot 500\ ps) / (3 \cdot 250\ ps) = 1{,}33$$

El rendiment del nou disseny B és 1,33 vegades més ràpid que el de l'original.

EXEMPLE 6: Suposem un computador A, amb f_A = 2GHz, que executa un programa en t_{exe} = 10s. Amb certes millores al circuit, s'executa en 6s, però augmenta el número de cicles en un factor 1,2. ¿Quina freqüència de rellotge té el nou disseny?

Solució: Siguin t_A, t_B els temps d'execució, f_A, f_B les freqüències de rellotge i n_A, n_B el número de cicles. Tenim dues equacions amb dues incògnites: n_A i f_B

$$t_A = n_A/f_A \quad \text{i} \quad t_B = n_B/f_B = (1{,}2 \cdot n_A)/f_B$$

Aïllem $n_A = t_A \cdot f_A$ en la primera equació i la substituïm en la segona

$$t_B = 1{,}2 \cdot (t_A \cdot f_A)/f_B$$

Aïllem f_B i calculem el resultat

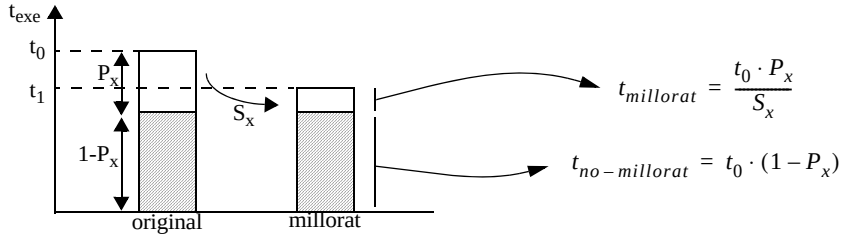
$$f_B = (1{,}2 \cdot t_A \cdot f_A)/t_B = (1{,}2 \cdot 10\text{s} \cdot 2\text{GHz})/6\text{s} = 4\text{GHz}$$

1.8

3. Llei d'Amdahl[4]

"*El màxim guany (speedup) que es pot aconseguir minimitzant el retard d'una part (d'un circuit, d'un programa etc.) està limitat per la fracció de temps que representa aquesta part sobre el temps total*".

Suposem per exemple un programa que s'executa en un temps t_0. Si optimitzem una subrutina, que comporta una fracció P_x del temps total, i aconseguim que aquesta subrutina tingui un guany de rendiment S_x, el temps d'execució total es redueix de t_0 a t_1.



Quin ha estat el guany de rendiment total S?

$$S = \frac{t_0}{t_1} = \frac{t_0}{\frac{t_0 \cdot Px}{Sx} + t_0 \cdot (1 - Px)} = \frac{1}{\frac{Px}{Sx} + (1 - Px)}$$

Quin guany total màxim S_{max} es pot aconseguir amb la millor optimització possible d'aquesta subrutina (suposant un guany S_x infinit)?

$$\lim_{Sx \to \infty} S = S_{max} = \frac{1}{1 - Px}$$

Exemple: Si millorem una part del programa, que abarca un 80% del temps, podem aspirar com a màxim a un guany total de:

$$S_{max} = 1/ (1 - 0{,}80) = 5$$

Conclusió: Cal concentrar els esforços d'optimització en aquelles parts del programa (o circuit) que comporten el major cost en temps.

4. Gene M. Amdahl. "Validity of the single processor approach to achieving large scale computing capabilities". AFIPS Spring Joint Computer Conference,1967.

1.5

4. Mesures de dissipació de potència

El 1965 en Gordon Moore (co-fundador d'Intel) va predir que la densitat de transistors al chip es podria duplicar cada 2 anys (a la pràctica, cada 18 mesos). Degut a la miniaturització, que permet reduir el temps de conmutació dels transistors, la freqüència de rellotge ha anat creixent i en conseqüència la dissipació de potència ha crescut en paral·lel (Fig 1.15 del P&H). L'energia consumida es converteix de la forma elèctrica a la forma de calor i cal dissipar-lo de manera forçada amb refrigeració per evitar que la temperatura creixi per sobre dels llindars de tolerància del circuit. Però ha arribat el punt que no són possibles dissipadors més potents en alguns computadors de cost limitat, i la corba històrica d'augment de freqüència comença a estancar-se. A això cal afegir-hi la gran demanda de dispositius mòbils que disposen d'un espai molt limitat per a dissipadors, i que a més necessiten un molt baix consum ja que s'alimenten per bateries.

La potència és l'energia dissipada per la circulació de corrents elèctrics en el circuit, i es pot dividir en dos tipus: la potència dinàmica (P_d), causada pels corrents de càrrega i descàrrega durant la conmutació de les portes; i la potència estàtica (P_s), fruit dels corrents paràsits que circulen pels transistors independenment de si conmuten o no

$$\mathbf{P = P_d + P_s}$$

4.1 Càlcul de la potència dinàmica (P_d)

El consum d'energia dinàmic és l'originat pels corrents de càrrega i descàrrega de les capacitats equivalents dels transistors que tenen lloc en les conmutacions (vegeu el requadre de la pàgina següent, per a més detalls). En cada cicle complet de càrrega i descàrrega d'una porta es produeix un corrent elèctric ja que hi ha un desplaçament net de càrrega de Vcc al terra. La càrrega desplaçada en un cicle complet és

$$q_i = C_i \cdot V$$

On C_i és la capacitat equivalent en farads del circuit connectat a la sortida de la porta. C_i depèn del nombre (fan-out), material i grandària dels transistors i connectors connectats a la sortida de la porta. V és la diferència de tensió en volts entre l'estat inicial i final dels elements capacitius, igual a Vcc. El corrent elèctric generat en tot el circuit durant 1 cicle de rellotge és igual a la càrrega i descàrrega agregada de totes les portes que conmutin dividida pel temps de cicle de rellotge t_c

$$I = \Sigma\, q_i/t_c = \Sigma\, C_i \cdot V \cdot f_{clock}$$

No toes les portes conmuten en cada cicle de rellotge. Es defineix el *factor d'activitat* (α) com el promig de portes que conmuten globalment en tot el circuit, per a un determinat programa o conjunt de programes representatius, de manera que $\Sigma\, C_i = \alpha \cdot C$, essent C la capacitància agregada de tot el circuit. Per tant, la potència dinàmica dissipada (consum d'energia per unitat de temps) en watts pel circuit és

$$\mathbf{P = \alpha \cdot C \cdot V^2 \cdot f_{clock}}$$

Per tenir una idea, són bastant comuns els valors de α pròxims a 0,1.

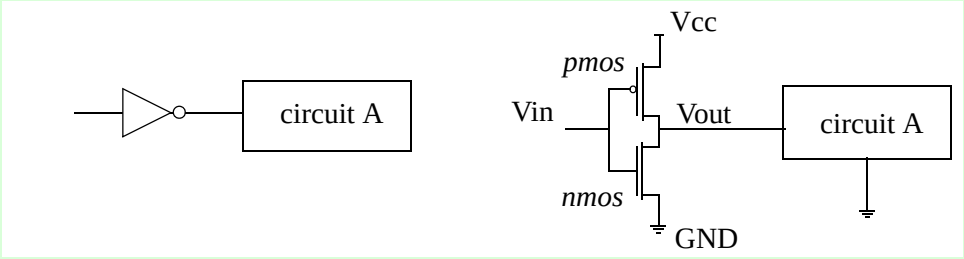
L'energia dissipada en un temps t, en Joules, és

$$\mathbf{E = P \cdot t}$$

.

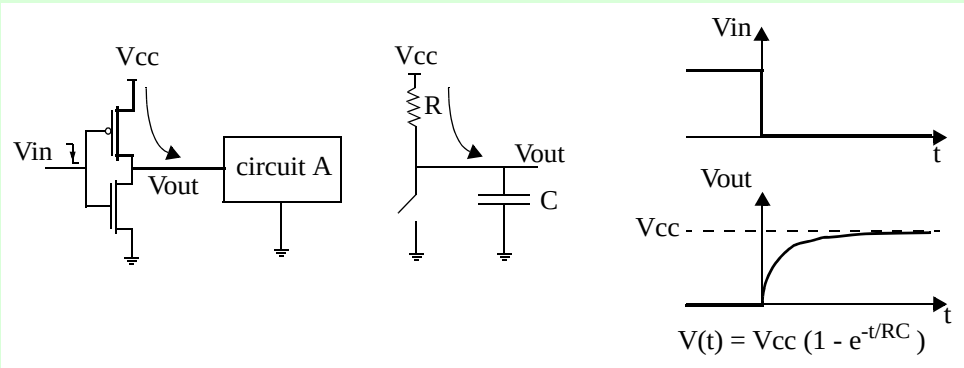
Cicle de càrrega/descàrrega en el circuit equivalent RC d'una porta lògica

Les portes d'un circuit estan majoritàriament construïdes amb transistors de la familia CMOS. Per exemple, una porta NOT:

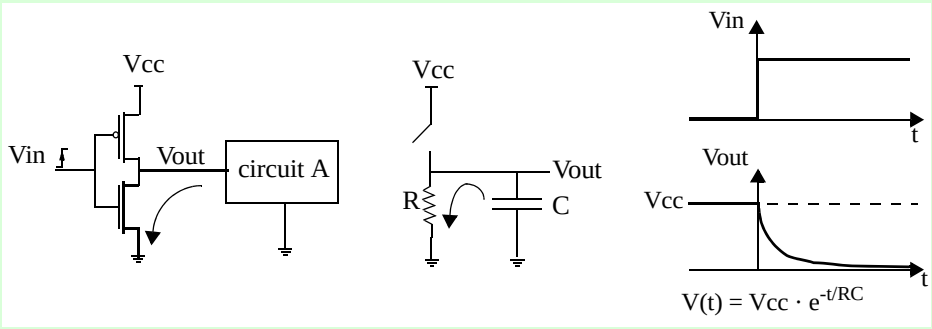


El transistor *pmos* condueix (curt-circuit) quan la tensió d'entrada baixa de cert llindar (threshold) Vt, i roman tallat (circuit-obert) si la tensió és superior. A la inversa, el transistor *nmos* condueix quan la tensió d'entrada puja per sobre de cert llindar Vt, i resta obert si la tensió és inferior. Aquests llindars determinen els nivells lògics 1 i 0.

Càrrega: Quan la tensió a l'entrada de la porta Vin passa del valor lògic 1 al 0 (flanc descendent), el transistor pmos passa a conduir i el nmos queda obert. Al conduir, el corrent flueix a través de la seva resistència equivalent R, i la càrrega elèctrica s'acumula en les etapes d'entrada del circuit A (en les portes d'altres transistors), que actuen així a manera d'un condensador C. Veiem el circuit equivalent i la gràfica de tensió:



Descàrrega: Quan la tensió a l'entrada de la porta Vin passa del valor lògic 0 a 1 (flanc ascendent), el transistor nmos passa a conduir i el pmos queda obert. Al conduir, el corrent flueix a través de la resistència equivalent R, i la càrrega elèctrica prèviament acumulada en les etapes d'entrada del circuit A (condensador C) es descarrega cap al terra. Veiem el circuit equivalent i la gràfica de tensió:



4.2 Càlcul de la potència estàtica (P_s)

El consum dinàmic d'energia no és l'únic que té lloc en el circuit. Hem suposat que pels transistors en circuit-obert no hi circula corrent, però en realitat no estan mai completament en circuit-obert sinó que sempre hi circula de manera constant un petit corrent paràsit anomenat corrent fuga I_{leak}.

$$\mathbf{P_s = I_{leak} \cdot V}$$

Aquest corrent no depèn de l'activitat del processador sinó del nombre total de transistors. Amb la miniaturització dels transistors també disminueix la tensió llindar V_t, i produeix un augment exponencial dels corrents de fuga i de la corresponent potència estàtica. Per a V_t = 0,2V la potència estàtica podria arribar a ser el 50% del total!

Conseqüències derivades de la miniaturització

- **C**: Transistors cada cop més petits i amb menor capacitat. Però com que n'hi caben molts més, la capacitat agregada total C no ha variat significativament.
- **f**: La reducció del temps de conmutació ha permès augmentar la freqüència de rellotge f per augmentar el rendiment (s'ha multiplicat per 300 en 25 anys, Fig 1.15 del P&H). Això comporta un increment proporcional de potència dinàmica.
- **V**: Per compensar l'augment de potència causat per l'augment de la freqüència f, s'ha reduït V progressivament (de 5 a 1V), i per tant també Vt de manera acompassada (de 0,7 a 0,4V). Aquesta estratègia ha estat molt efectiva per reduir la potència dinàmica, ja que la influència de V és quadràtica. Però la reducció de Vt provoca un augment exponencial dels corrents de fuga, i de la consegüent potència estàtica, de manera que la reducció de V no permet mantenir la potència dissipada constant (s'ha multiplicat per 30 en 25 anys, Fig 1.15 del P&H).

El resultat global dels tres factors és que la freqüència f ja no pot augmentar-se més, a risc d'augmentar el consum d'energia més enllà de la capacitat de dissipació de calor, pel risc que la temperatura sobrepassi els marges de tolerància.

Algunes tècniques de reducció del consum

- ***Clock gating***: permet reduir α. Consisteix a inhabilitar selectivament la conmutació del senyal de rellotge de determinats circuits durant períodes de temps en què sabem que els biestables d'aquest circuit no han de canviar d'estat.
- ***Dynamic voltage and frequency scaling*** **(DVFS)**: Permet reduir V i f. Consisteix a reduir temporalment la tensió i la freqüència d'una part del circuit quan aquesta no necesita operar a la màxima velocitat, ja que això fa que siguin més lents però consumeixin molt menys.
- ***Power gating***: Permet reduir tant el consum dinàmic com l'estàtic tallant temporalment l'alimentació de determinats circuits quan no s'han d'usar durant un llarg període de temps. Per exemple, deshabilitant alguns nuclis del processador, o alguns bancs de la cache. Però cal tenir en compte que tots els elements de memòria perden el seu contingut. A més a més, no és un recurs molt flexible ja que els circuits de control addicionals ocupen una àrea significativa i a més l'activació i desactivació requereix un temps transitori significatiu.