Introducció als Computadors

Tema 14: Llenguatge *assembler* SISA

http://personals.ac.upc.edu/enricm/Docencia/IC/IC14.pdf

Enric Morancho
(enricm@ac.upc.edu)

Departament d'Arquitectura de Computadors
Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

2020-21, 1^{er} quad.

Presentació publicada sota llicència Creative Commons 4.0

Índex

Full de ruta

	Tema							
	7	8	9	10	11	12	13	14
Unitat de Control	UCE	UCE	UCE	UCG	UCG	UCG	UCG	UCG
Unitat de Procés	UPE	UPG	UPG	UPG	UPG	UPG	UPG	UPG
Entrada/Sortida	-	-	IO	IO	IO	IO	IO	IO
Memòria RAM	-	-	-	-	MEM	MEM	MEM	MEM
Harvard unicicle	-	-	-	-	-	✓	-	-
Harvard multicicle	-	-	-	-	-	✓	-	-
Von Neumann	-	-	-	-	-	-	✓	✓
Lleng. *assembler*	-	-	-	✓	✓	✓	✓	✓

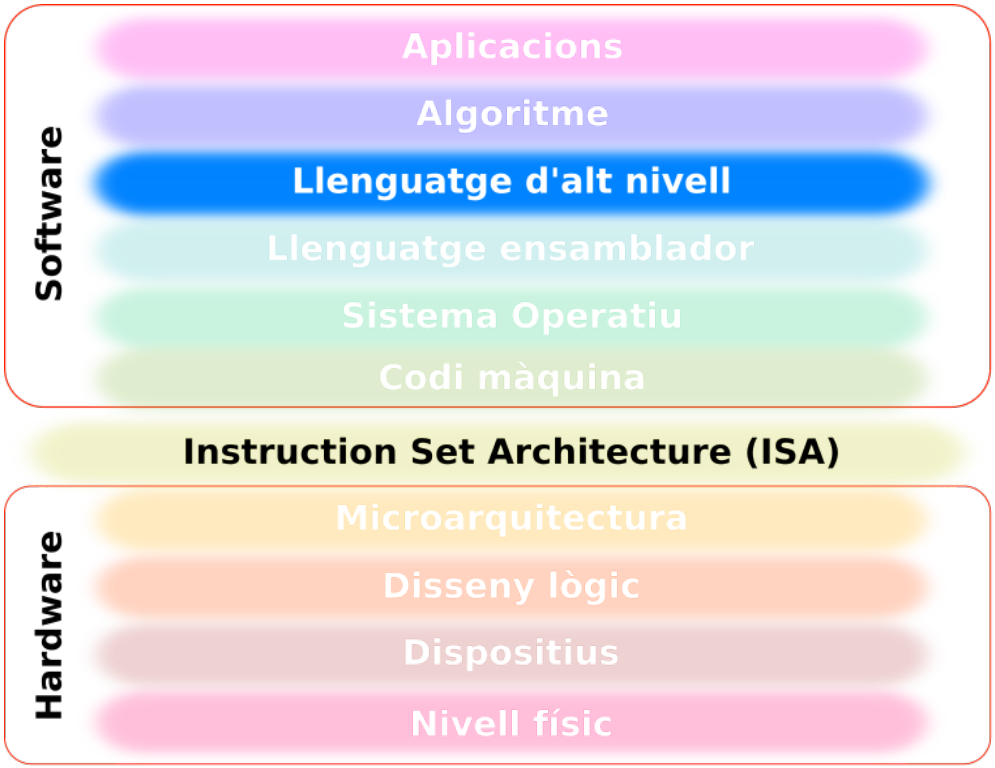
Nivells de llenguatges de programació

Nivells de llenguatges de programació


```
1009  bad_gs:
1010          SWAPGS                                  /* switch back to user gs */
1011  .macro ZAP_GS
1012          /* This can't be a string because the preprocessor needs to see it. */
1013          movl $__USER_DS, %eax
1014          movl %eax, %gs
1015  .endm
1016          ALTERNATIVE "", "ZAP_GS", X86_BUG_NULL_SEG
1017          xorl    %eax, %eax
1018          movl    %eax, %gs
1019          jmp     2b
1020          .previous
1021
1022  /* Call softirq on interrupt stack. Interrupts are off. */
1023  ENTRY(do_softirq_own_stack)
1024          pushq   %rbp
1025          mov     %rsp, %rbp
1026          ENTER_IRQ_STACK regs=0 old_rsp=%r11
1027          call    __do_softirq
1028          LEAVE_IRQ_STACK regs=0
1029          leaveq
1030          ret
1031  ENDPROC(do_softirq_own_stack)
```

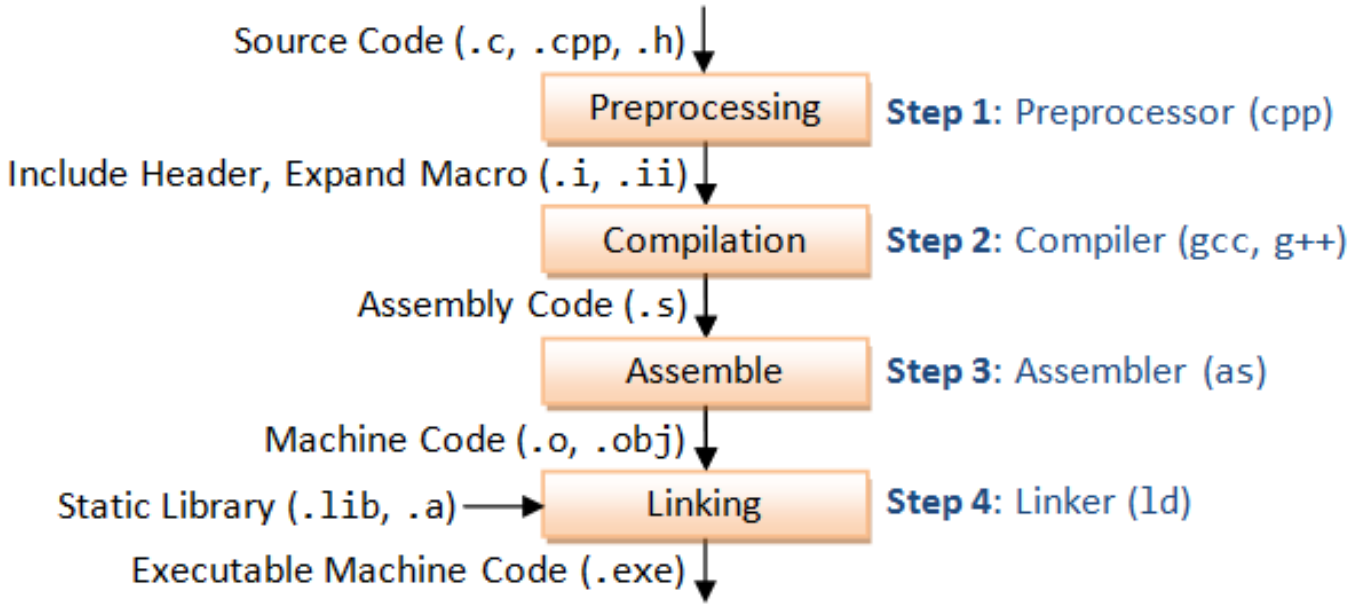
Representació simbòlica del codi màquina

Nivells de llenguatges de programació



Generació del programa en codi màquina

- Passos a seguir:
  - Dependrà del llenguatge de programació en què treballeu
  - Des d'un llenguatge d'alt nivell, identifiquem 4 etapes:
    - Preprocessament, compilació, ensamblatge, muntatge
    - El "compilador" `gcc`, `g++` s'encarrega de tot

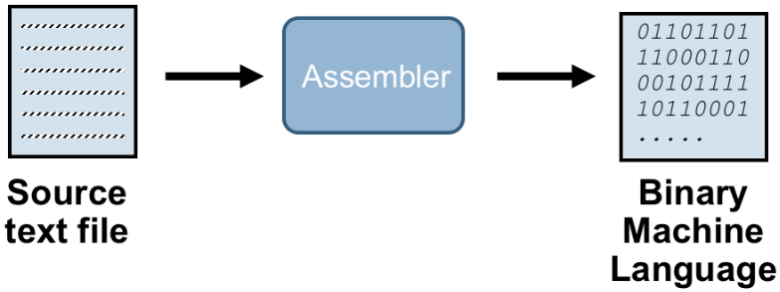


[1]

  - Com a resultat, es genera el "fitxer executable"
- A IC anirem directament a l'step 3 (*assembler*)
  - Programem en llenguatge ensamblador!
  - Obviem l'step 4 (*linking*)

Motivació llenguatge *assembler*

- Programar directament en codi màquina és inhumà
- El llenguatge *assembler* ens permet "humanitzar" el codi màquina
  - Versió *human friendly* del codi màquina
  - `1000110111111110` versus `BNZ R6, -2`
- Caldrà traduir el codi escrit en llenguatge *assembler* (*source text file*) a codi màquina (*binary, object code*)
  - Ho farà un programa anomenat *assembler*
    - *assembler* fa referència tant al llenguatge de programació com al programa que tradueix a codi màquina!!!



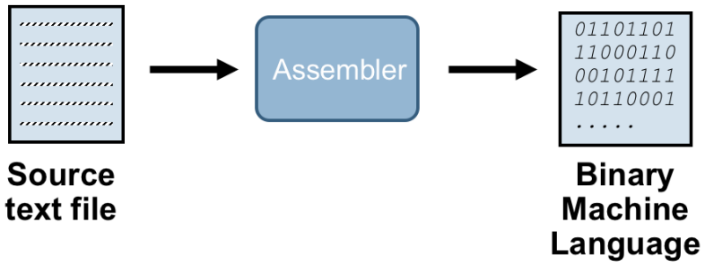
[2]

- Veurem funcionalitats del llenguatge *assembler* que ens ajudaran a:
  - Ser més productius
  - Cometre menys errors
  - Automatitzar tasques

Índex

Codi font *assembler*

- El codi font es processa línia a línia



[2]

- Al codi font hi podem trobar:
  - Instruccions SISA
    - Amb el format vist fins ara: mnemotècnic + registres/constants
    - Traduïdes als 16 bits de la codificació SISA
  - Comentaris
    - Comencen amb el caràcter `;` i acaben a final de línia
    - S'ignoren al generar el fitxer amb codi màquina
  - Etiquetes
    - Cadena alfanumèrica acabada amb `:`
    - Representa l'adreça de memòria que correspon a la resta de la línia
  - Directives del llenguatge *assembler*
    - Paraules clau que comencen per `.` i, si s'escau, tenen paràmetres
  - Crides a funcions del llenguatge *assembler*
    - `hi()`, `lo()`

Seccions

- A un codi font *assembler* trobarem instruccions i dades
  - Les instruccions s'agrupen en una o vàries seccions de codi
    - Contenen les instruccions SISA
    - Comencen amb la directiva `.text`
  - Les dades s'agrupen en una o vàries seccions de dades
    - Contenen la reserva d'espai de memòria i la seva inicialització
    - Comencen amb la directiva `.data`
- El codi font *assembler* conté una seqüència de seccions
  - L'inici d'una secció comporta la finalització de la secció anterior
- La directiva `.end` indica el final de la darrera secció i el final del codi font
  - El contingut posterior del fitxer font és ignorat

Etiquetes

- És una cadena alfanumèrica seguida del caràcter :
- Poden aparèixer tant a seccions de codi com de dades
- Permet identificar de forma simbòlica una adreça de memòria
  - Podrem fer referència a adreces de memòria que encara no són conegudes
- Exemple:

```
      LD   R1, 0(R3)
      BZ   R1, et1      ; Saltem a etiqueta et1
      ADD  R2, R0, R1
      ADD  R3, R1, R2
et1:  AND  R1, R2, R3 ; et1 representa adreça de la instr
```

  - L'*assembler* calcularà el desplaçament corresponent al BZ
    - En aquest cas, +2
  - El programador es despreocuparà de fer aquest càlcul
  - Codi font més llegible i fàcil de mantenir

Reserva d'espai i inicialització de dades

- El llenguatge *assembler* ofereix directives per a dimensionar i inicialitzar les seccions de dades
  - `.space` *size*, *fill*
    Reserva *size bytes* consecutius i els inicialitza amb el *byte* *fill*. El paràmetre *fill* és opcional; si no hi és, s'inicialitzarà amb el *byte* 0
  - `.byte` *fill-1*, *fill-2*, ...,*fill-n*
    Reserva i inicialitza *n bytes* consecutius amb els valors *fill-1*, *fill-2*, ..., *fill-n*
  - `.word` *fill-1*, *fill-2*, ...,*fill-n*
    Reserva i inicialitza *n words* consecutius amb els valors *fill-1*, *fill-2*, ..., *fill-n* (*byte* de menys pes a l'adreça parell)
  - `.even`
    Si volem tenir la garantia que la següent sentència/directiva del codi font *assembler* s'ubiqui a una adreça parell, aquesta directiva inserta, si cal, una directiva `.byte`

Reserva d'espai i inicialització de dades

- Exemple:

```
.data
v:  .space      15, 20        ; 15 bytes inicialitzats a 20
    .even
w:  .word       0x178A
    .byte       12, 0xFA, -1
    .even                     ; Inserta un byte
z:  .word       0xABCD
```

  - El primer `.even` inserta un *byte* per garantir que `.word` estigui ben alineat, a una adreça parell
  - El segon `.even` inserta un *byte* perquè hem declarat 3 *bytes* després del *word*
  - Si les dades es carreguem a partir de 0x3000 llavors `v`=0x3000, `w`=0x3010, `z`=0x3016
    - Si fos "`.space 16, 20`", el resultat seria idèntic

Definició de constants numèriques

- Dues possibles sintaxis:
  - `nom_constant = valor`
  - `.set nom_constant, valor`
- Exemple:

```
          Mida = 100
.data
vector:  .space Mida        ; Reserva "Mida" bytes
```

  - Associa al símbol `Mida` el valor 100
  - A partir d'aquest moment, l'*assembler* substituirà totes les aparicions del símbol `Mida` pel valor 100
    - **La definició d'una constant no ocuparà espai a memòria**
    - Seria equivalent a un "Buscar i reemplaçar totes" al codi font o a un `#define` de C/C++
- Avantatges:
  - Codi més llegible
  - Facilita el manteniment de codi
    - Si el programador canvia el valor de la constant, l'*assembler* propagarà el canvi a tots els llocs on es referencia

Funcions hi() i lo()

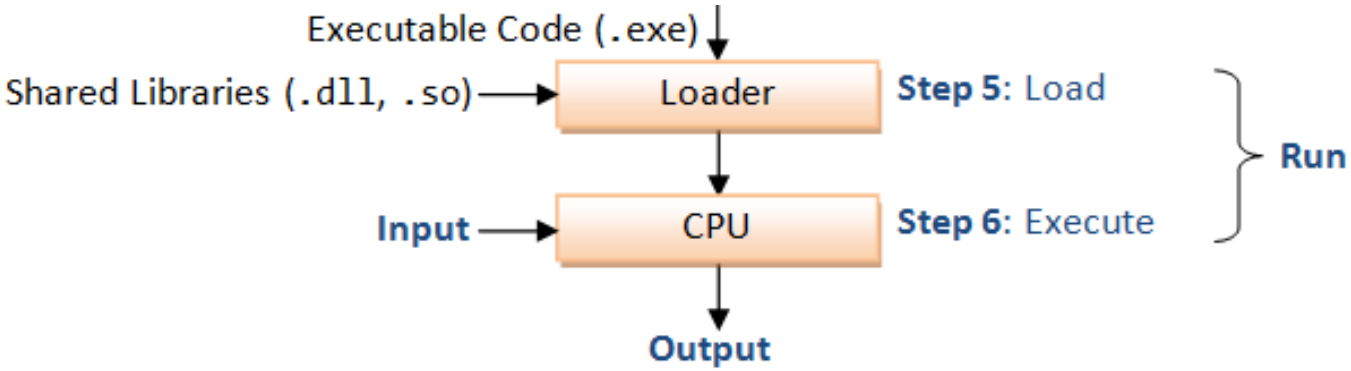
- Permeten obtenir la part alta/baixa d'una dada de 16 bits
  - Siguin adreces de memòria, etiquetes, o constants numèriques
- Estalvia al programador haver de fer el càlcul
- Exemple:
  - Assumim que la secció `.data` es carrega a partir de `0xCAFE`

```
        N = 24335                   ; 24335 = 0x5EA1

.data
        space 2                     ; 0xCAFE
vector: space 100, 0xFF             ; 0xCB00 (0xCAFE+2)

.text
        MOVI   R0, lo(N)            ; lo(N) = 0xA1
        MOVHI  R0, hi(N)            ; hi(N) = 0x5E

        MOVI   R1, lo(vector)       ; lo(vector) = 0x00
        MOVHI  R1, hi(vector)       ; hi(vector) = 0xCB
```

Índex

Executar un programa en codi màquina

- Invocar l'executable, fer doble click sobre l'icona,...
  - Feina del *loader*
    - Part del Sistema Operatiu
    - Llegeix el fitxer generat per l'*assembler* i el carrega a memòria
    - També haurà d'inicialitzar el registre `PC`



[3]

- A les pràctiques d'IC ho feu a mà a l'inicialitzar la memòria
  - Inicialitzeu cada mòdul de memòria per separat

Finalització de l'execució del programa

- El nostre computador no para mai !!!
- Al final del codi hauria d'haver-hi una invocació al Sistema Operatiu per notificar la finalització del programa
  - El *loader* podrà carregar un nou programa
- A IC no tenim Sistema Operatiu
  - Si el programa no és un bucle, després de la darrera instrucció es passarà a executar PC+2 i el contingut d'aquella posició de memòria s'interpretarà com a una instrucció SISA,...

SISC Von Neumann versus PC domèstic

- Al PC de casa/mòbil tinc varis programes executant-se alhora!
  - És un ordinador complet i té un sistema operatiu amb cara i ulls
    - En particular, implementa interrupcions (les veureu a EC)
  - Tots els programes estan carregats a memòria alhora
  - El Sistema Operatiu, uns 1000 cops per segon, decideix quin dels programes carregats pot continuar la seva execució
    - Tot i que el computador només tingui un processador, l'usuari té la sensació que tots els programes s'executen alhora
    - Cal garantir que un programa no escrigui a porcions de memòria que no li toquen (EC, SO)
    - Cal guardar a memòria els valors del registre PC, i els dels banc de registres cada cop que s'atura l'execució d'un programa
    - Cal assignar valors al registre PC i als del banc de registres cada cop que es reprèn l'execució d'un programa

Índex

Exercici típic

- Assumint que la secció de dades es carrega a `0xA000` i a continuació la de codi, indiqueu el valor de l'etiqueta `vector` i el contingut de l'adreça `0xA06C` del següent programa?

```
          N = 24335

.data
          space 2
vector:   space 100, 0xFF

.text
          MOVI    R0, lo(N)
          MOVHI   R0, hi(N)

          MOVI    R1, lo(vector)
          MOVHI   R1, hi(vector)
```

Exercici típic

- Assumint que la secció de dades es carrega a 0xA000 i a continuació la de codi, indiqueu el valor de l'etiqueta `vector` i el contingut de l'adreça 0xA06C del següent programa?

```
                    N = 24335

                    .data
0xA000              space 2
0xA002      vector: space 100, 0xFF

                    .text
0xA066              MOVI   R0, lo(N)
0xA068              MOVHI  R0, hi(N)

0xA06A              MOVI   R1, lo(vector)
0xA06C              MOVHI  R1, hi(vector)
```

- `vector = 0xA002`
- $Mem_w[0xA06C] = 0x93A0$

El programa ensamblador de la derecha se ha traducido a lenguaje máquina para ser ejecutado en el SISC Von Neumann, situando la sección `.data` a partir de la dirección `0xA000` de memoria y justo a continuación la sección `.text`.

a) Una vez cargado el programa en memoria:

- ¿A qué dirección de memoria corresponden las etiquetas, o direcciones simbólicas, `L1` y `V2`? (0,5 puntos)

`L1 = 0x`	`V2 = 0x`

- ¿Cuál es el word almacendo en las siguientes direcciones de memoria? (0,5 puntos)

Mem_W`[0xA010] = 0x`
Mem_W`[0xA02A] = 0x`

b) Una vez ejecutado el programa en el computador SISC Von Neumann ¿Cuál es la dirección de memoria escrita por la instrucción `ST`?¿Cuál es el valor escrito? (0,75 puntos)

Mem_W`[0x ] = 0x`

c) Indicad el número total de instrucciones que ejecuta el programa, así como cuántas son lentas y cuántas son rápidas (0,25 puntos)

$N_{total} =$	$N_{lentas} =$	$N_{rápidas} =$

```
.data
V1:     .word 1, 2, 4, 8
        .word 16, 32, 64
        .word -1
V2:     .byte 7, 6, 5, 4
        .byte 3, 2, 1
        .even
V3:     .word 0

.text
        MOVI  R0, lo(V1)
        MOVHI R0, hi(V1)
        MOVI  R1, lo(V2)
        MOVHI R1, hi(V2)
        MOVI  R2, 0
        MOVI  R3, 0xFF
L1:     LD    R4, 0(R0)
        CMPEQ R5, R3, R4
        BNZ   R5, L2
        LDB   R6, 0(R1)
        SHL   R4, R4, R6
        ADD   R2, R2, R4
        ADDI  R0, R0, 2
        ADDI  R1, R1, 1
        BZ    R5, L1
L2:     MOVI  R7, lo(V3)
        MOVHI R7, hi(V3)
        ST    0(R7), R2
.end
```

El programa ensamblador de la derecha se ha traducido a lenguaje máquina para ser ejecutado en el SISC Von Neumann, situando la sección `.data` a partir de la dirección `0xA000` de memoria y justo a continuación la sección `.text`.

a) Una vez cargado el programa en memoria:

- ¿A qué dirección de memoria corresponden las etiquetas, o direcciones simbólicas, `L1` y `V2`? (0,5 puntos)

`L1 = 0xA026`	`V2 = 0xA010`

- ¿Cuál es el word almacendo en las siguientes direcciones de memoria? (0,5 puntos)

Mem_W`[0xA010] = 0x0607`
Mem_W`[0xA02A] = 0x8B06`

b) Una vez ejecutado el programa en el computador SISC Von Neumann ¿Cuál es la dirección de memoria escrita por la instrucción `ST`?¿Cuál es el valor escrito? (0,75 puntos)

Mem_W`[0xA018] = 0x0380`

c) Indicad el número total de instrucciones que ejecuta el programa, así como cuántas son lentas y cuántas son rápidas (0,25 puntos)

$N_{total} = 75$	$N_{lentas} = 16$	$N_{rápidas} = 59$

```
.data
V1:     .word 1, 2, 4, 8
        .word 16, 32, 64
        .word -1
V2:     .byte 7, 6, 5, 4
        .byte 3, 2, 1
        .even
V3:     .word 0

.text
        MOVI  R0, lo(V1)
        MOVHI R0, hi(V1)
        MOVI  R1, lo(V2)
        MOVHI R1, hi(V2)
        MOVI  R2, 0
        MOVI  R3, 0xFF
L1:     LD    R4, 0(R0)
        CMPEQ R5, R3, R4
        BNZ   R5, L2
        LDB   R6, 0(R1)
        SHL   R4, R4, R6
        ADD   R2, R2, R4
        ADDI  R0, R0, 2
        ADDI  R1, R1, 1
        BZ    R5, L1
L2:     MOVI  R7, lo(V3)
        MOVHI R7, hi(V3)
        ST    0(R7), R2
.end
```

Índex

Conclusions

- El llenguatge *assembler* facilita la vida al programador de "baix nivell"
  - Codi més llegible
  - Codi més fàcil de mantenir
  - Ajuda a evitar alguns errors
    - Fa la feina "bruta" (càlcul de desplaçaments, parts altes, baixes....)
- Els llenguatges *assembler* d'altres ISA's ofereixen funcionalitats similars a les vistes a l'*assembler* SISA

Referències I

Llevat que s'indiqui el contrari, les figures, esquemes, cronogrames i altre material gràfic o bé han estat extrets de la documentació de l'assignatura elaborada per Juanjo Navarro i Toni Juan, o corresponen a enunciats de problemes i exàmens de l'assignatura, o bé són d'elaboració pròpia.

[1] [Online]. Available: https://medium.com/@vietkieutie/what-happens-when-you-type-gcc-main-c-2a136896ade3.

[2] [Online]. Available: https://computationstructures.org/notes/assembly/notes.html.

[3] [Online]. Available: https://codingmeta.com/create-a-program-c/.

Introducció als Computadors

Tema 14: Llenguatge *assembler* SISA

http://personals.ac.upc.edu/enricm/Docencia/IC/IC14.pdf

Enric Morancho
(enricm@ac.upc.edu)

Departament d'Arquitectura de Computadors
Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

2020-21, 1^{er} quad.

Presentació publicada sota llicència Creative Commons 4.0