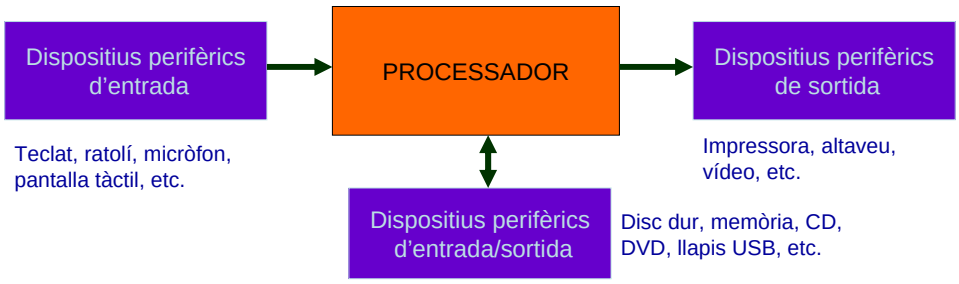
Fonaments de Computadors: introducció amb AVR (I)

Joan Oliver
Departament de Microelectrònica i Sistemes Electrònics
Universitat Autònoma de Barcelona
(Joan.Oliver@uab.es)

Índex

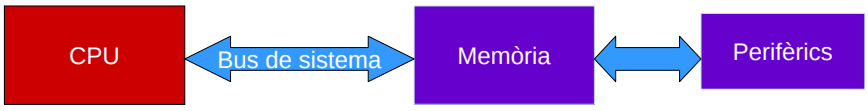
Arquitectura genèrica d'un computador

- Un processador és un component digital capaç d'interpretar instruccions (emmagatzemades en memòria) de forma ordenada, de processar dades i de generar informació de sortida.
- És un sistema complex, agrupació de subsistemes



Arquitectura genèrica d'un computador (II)

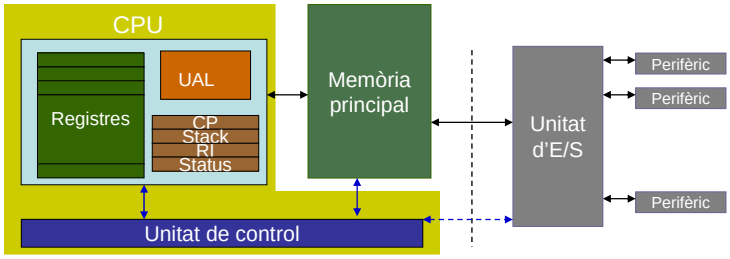
- El processador consta de
  - CPU. Unitat central de procés capaç de seqüenciar operacions, de processar dades i de donar resultats.
  - Memòria. Unitat d'emmagatzemament d'informació: programa i dades.



- Funcionament del processador
  - La CPU processa les dades que procedeixen de la memòria i en retorna el resultat
  - El processador i la memòria (principal) es comuniquen a través del bus de sistema
  - La velocitat de procés de la CPU és funció de les velocitats de memòria i de bus de sistema
  - Velocitats actuals del rellotge de bus de sistema estan sobre el GHz
  - Per accelerar la velocitat de procés la CPU disposa de registres (memòria molt específica) a prop i de memòria cau (més ràpida que la RAM estàndard)

Arquitectura von Neumann (I)

- L'arquitectura del computador en defineix el seu comportament funcional
- El model bàsic el va establir l'any 1945 von Neumann, arquitectura que encara avui en dia és vàlida en molts fabricants
- La màquina von Neumann és capaç d'executar un conjunt d'instruccions elementals, instruccions màquina que s'han de guardar en la memòria principal i que poden ser llegides i executades.
- Consta de les unitats funcionals
  - Memòria principal
  - Unitat Central de Procés (CPU)
    - Datapath: unitat aritmètico lògica i regsitres
    - Unitat de control
  - Unitat d'entrada/sortida (perifèrics)



Arquitectura von Neumann (II)

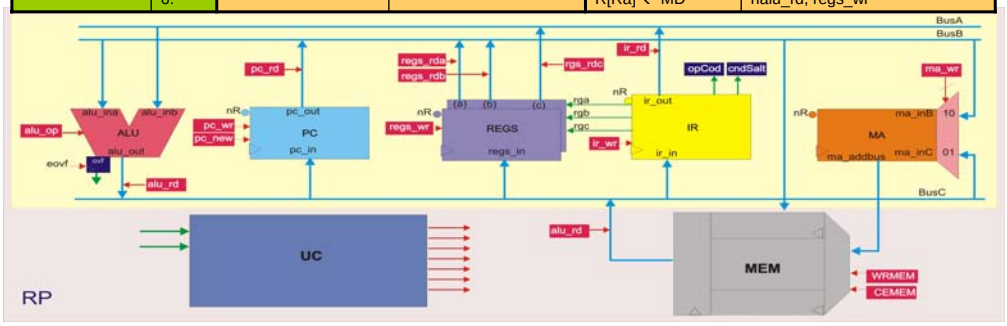
- La memòria principal està constituïda per un conjunt de cel·les idèntiques, d'un nombre fix de bits, organitzades de forma consecutiva que permeten ser adreçades a partir d'una adreça específica. Les instruccions i dades s'obtenen de la memòria a través del procés de lectura i s'hi guarden amb el procés d'escriptura. L'arquitectura von Neumann es caracteritza per compartir en el mateix espai de memòria programes i dades
- La UAL és la unitat que permet realitzar operacions concretes sobre dades. Les dades (operands) provenen de la memòria principal o de registres que guarden dades de forma temporal
- La unitat de control és l'encarregada de controlar (a través dels senyals de control) les accions que es realitzen en el processador: lectura de les instruccions de memòria, , execució de les instruccions i control del seqüenciament
- Al conjunt de unitat aritmètico/lògica, registres i unitat de control se l'anomena Unitat Central de Procés (CPU).
- La comunicació amb el computador es realitza a través dels perifèrics.
  - Els perifèrics són dispositius físics que permeten la comunicació amb el computador
    - ... comunicació home-màquina, en les que les persones interaccionen amb el computador
    - ... comunicació màquina-màquina, en les que els computadors interaccionen entre ells
    - ... comunicació màquina-home, en les que el computador interacciona amb les persones
  - Cada perifèric imposa els seus propis requisits. Per exemple, la comunicació del computador amb l'home es sol fer mitjançant terminals i impressores, fet que obliga a fer operacions de traducció donat que els llenguatges emprats són totalment divergents (el computador treballa a nivell digital).
  - Un altre aspecte a tenir en compta és la velocitat de transferència de dades que sol tenir un impacte gran en el funcionament del computador que normalment és molt més ràpid que no pas, per exemple, una impressora.
  - El maneig de la informació entre computador i perifèric pot ser feta pel mateix processador o per dispositius específics d'entrada/sortida que alleugereixen al processador d'aquestes tasques.

Arquitectura von Neumann (III)

Fases d'execució de les instruccions màquina

- La UC porta el control del *cicle d'execució d'instruccions.*
- Les fases d'execució d'un programa són:
  - Cerca de la instrucció a executar
    - El comptador de programes conté l'adreça de la instrucció a executar.
    - La UC envia a la memòria aquesta adreça per llegir la instrucció a executar.
    - La UC activa els senyals de control per carregar aquesta instrucció en el registre d'instruccions.
  - Descodificació de la instrucció i càlcul d'adreces dels operands
    - El registre d'instruccions analitza (descodifica) la instrucció i, si cal, llegeix els operands de la memòria principal.
    - Si cal, la UC activa els senyals de control per a la lectura, cerca i càrrega en registre dels operands.
  - Execució de l'operació
    - La UC s'encarrega de controlar l'operació sobre els operands i de guardar el resultat en registre o en memòria.
  - Increment del comptador de programes
    - Un cop executada la instrucció a executar es calcula l'adreça de la instrucció següent i, seguidament, es passa a la instrucció següent.
    - Actualment, molts processadors pre-calculen (automàticament) l'adreça de la instrucció següent un cop s'ha executat la cerca de la instrucció, fet que els permet estalviar-se una fase per instrucció.

	Cicle	ADD Ra, Rb, Rc	Control	LD Ra, Rb(c)	Control
		Ra ← Rb + Rc)		Ra ← R[Rb]+c	
Cerca	1:	MA ← PC PC ← PC+1	pc_rd,ma_wr=2 pc_nw	MA ← PC PC ← PC+1	pc_rd,ma_wr=2 pc_nw
	2:	MD ← Mem[MA]	CEmem	MD ← Mem[MA]	CEmem
Descodificació	3:	IR ← MD	mem_rd, ir_wr	IR ← MD	nalu_rd , ir_wr
Execució	4:	R[Ra] ← R[Rb]+R[Rc]	regs_rdb, regs_rdc, alu_op, alu_rd, regs_wr	MA ← R[Rb]+c	regs_rdb, ir_rd, alu_op, alu_rd, ma_wr=2
	5:			MD ← Mem[MA]	CEmem
	6:			R[Ra] ← MD	nalu_rd, regs_wr



Arquitectura d'un computador: concepte (I)

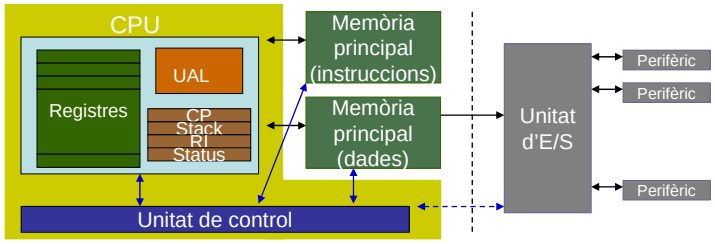
- Hi ha diferents taxonomies que intenten dividir el computador en un conjunt de capes en el que respecte al seu funcionament.
- Així, per exemple, la taxonomia de Levy que divideix el computador des del punt de vista funcional, estableix les següents capes:
  - Les microinstruccions. És la capa més baixa del computador i forma el nivell baix del hardware. Tot programa no és res mes que un conjunt de zeros i uns que, a través d'un control microprogramat, fan funcionar el computador. Forma el firmware.
  - Les instruccions màquina formen el segon nivell, el nivell alt del hardware. És el nivell d'interpretació que veu el programador d'assemblador, i que es considera la frontera entre software i hardware.
  - El sistema operatiu en forma el tercer nivell. Originàriament eren un conjunt de programes que ajudaven l'usuari en la interacció amb el computador. S'encarrega fonamentalment de l'administració dels recursos del computador.
  - En un quart nivell hi ha el conjunt de llenguatges de programació d'alt nivell (C/C++, Pascal, Ada, etc). Aquests llenguatges treballen a un nivell superior i, per compilació, generen un codi objecte, codi entre el sistema operatiu i el traductor.
  - Capa de llenguatge d'alt nivell. Forma el nivell més alt i el constitueixen el conjunt d'eines d'aplicació que corren sobre l'ordenador.

Arquitectura d'un computador: concepte (II)

- L'arquitectura d'un computador influeix de forma notable en la forma que un programador ha de crear un programa. En relació a la taxonomia establerta per Levy, correspon al nivell d'instruccions màquina i comprèn:
  - El joc d'instruccions màquina. Fa referència al conjunt d'instruccions, modes d'adreçament i formats de representació
  - Tipus i formats dels operands amb els que s'opera. Influeix en la precisió i resolució de dades i en les transformacions sobre representació de dades
  - Mapes de memòria i entrada/sortida. Per exemple, un punt principal de divergència entre les arquitectures Von Neumann i Harvard el forma el diferent nombre de memòries que composen cada arquitectura. L'arquitectura Von Neumann fa servir una única memòria per a guardar programes i daes. L'arquitectura Harvard empre, per a aquest menester, diferents memòries
  - La seqüència d'execució. Fa referència a cóm es seqüencia el programa (criden les instruccions). Normalment el control es realitza en base al comptador de programes, un apuntador a l'adreça de la instrucció que s'ha d'executar.

Arquitectura Harvard

- A diferència de l'arquitectura Von Neumann l'arquitectura Harvard
  - emmagatzema en memòries diferents programa i dades
  - els espais d'adreces de les memòries de programes i dades són diferents
  - les arquitectures de les memòries de programa i dades poden ser diferents
- L'avantatge és que en paral·lel es pot llegir instrucció i les dades, fet que li permet ser més ràpida
- En contra, cal tenir cura en el dimensionament de les memòries per evitar que cap d'elles quedi infra-utilitzada.
- És una arquitectura molt emprada en microcontroladors
- Tot i tenir memòries per a usos diferents, el principi de funcionament és similar al de l'arquitectura Von Neuman



Memòria

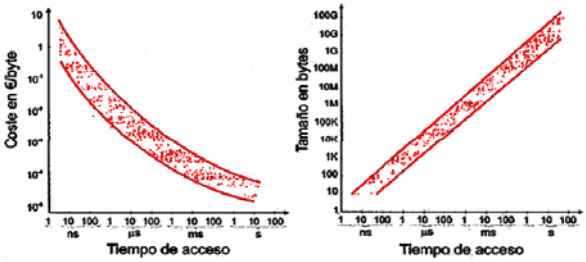
- Els programes i les dades s'emmagatzemen en memòria. Per a què un computador pugui executar un programa, aquest i les dades s'han de trobar en memòria principal
- Per accedir a una dada en memòria cal indicar la seva posició o adreça
- La memòria s'estructura en paraules: nombre de bits que es llegeixen/ escriuen en cada posició. Per tant, sobre la memòria es poden executar les accions de lectura i escriptura
- L'espai adreçable és el nombre de posicions de memòria que es pot adreçar, i depèn del nombre de bits que té l'adreça
- La capacitat de la memòria és el nombre de posicions de memòria que té
- Es parla de mapa de memòria com l'espai adreçable total pel computador. És a dir, si un computador empra un bus d'adreces de 16 bits, el seu mapa de memòria té una capacitat de 2^16 paraules.
- En casos necessaris es pot ampliar el mapa de memòria concatenant línies externes d'adreçament amb el bus d'adreces.

- Exemple: memòria de 1024 paraules de 8 bits
  - Paraula → 8 bits
  - Espai adreçable → 1024 posicions → #bits de l'adreça = 10 bits

Adreça	Contingut
0:	10011000
1:	00011000
2:	10000000
	10001000
...	...
1022:	10010000
1023:	00001000

Memòria: jerarquia de memòria (I)

- Les diferents tecnologies que s'empren en la fabricació de memòries i els usos a què aquestes es destinen es basen en tres propietats fonamentals:
  - El cost per bit
  - La capacitat d'emmagatzemament
  - El temps d'accés



- Les prestacions elevades dels computadors exigeixen gran quantitat de memòria que, al mateix temps, necessitaria ser de baix cost per a produir un producte competitiu.
- De la figura es demostra que són dos requeriments enfrontats → Cal establir una jerarquia de memòria.

Memòria: jerarquia de memòria (II)

-

Jerarquía de Memorias

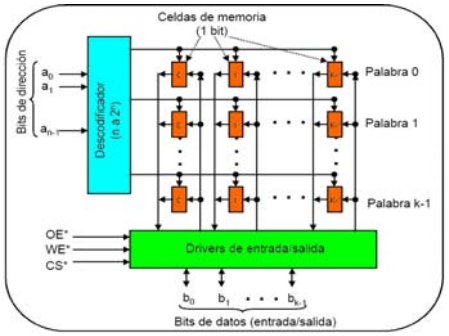
Jerarquía	Capacidad en bytes	Tiempo de acceso	Ancho de banda	Tipo	Acceso elemental
Registros	6-200	0,25-0,5 ns	20-100 GB/s	Biestables	Palabra
Mem. cache	8K-8M	0,5-25 ns	5-10 GB/s	RAM	Palabra
Mem. principal	1M-16G	80-200 ns	1-5 GB/s	RAM	Palabra
Mem. expandida	128M-20G	100-300 ns	1-5 GB/s	RAM	Varias Palabr.
Disco magnético	5G-100G	5-20 ms	20-150 MB/s	A. Directo	Sector
Cinta magnética	300K-800G	minutos	1-5 MB/s	A. Secuencial	Registro

Características de la jerarquía de memorias

Memòria: organització

- La memòria interna del computador sol estar formada per les primeres capes de la jerarquia:
  - Registres, normalment associats a la CPU i construïts en el mateix processador
  - Memòria principal, on resideixen les dades i els programes
  - I la memòria cau que és una memòria auxiliar emprada per accelerar l'accés a la memòria principal. Per a què un computador pugui executar un programa, aquest i les dades s'han de trobar en memòria principal.

- La memòria principal sol ser tota implementada a partir de memòria semiconductora.
- Segons la tecnologia les memòries poden ser de lectura/escriptura (SRAM, DRAM), o de lectura (ROM, PROM, E(E)PROM, Flash).
- Els xips de memòria semiconductora, tot i tenir diferents organitzacions, tots es basen en una arquitectura d'emmagatzemament en cel·les de memòria, adreçades per un bus d'adreces, amb drivers d'entrada/sortida, i la lògica de control de lectura/escriptura.



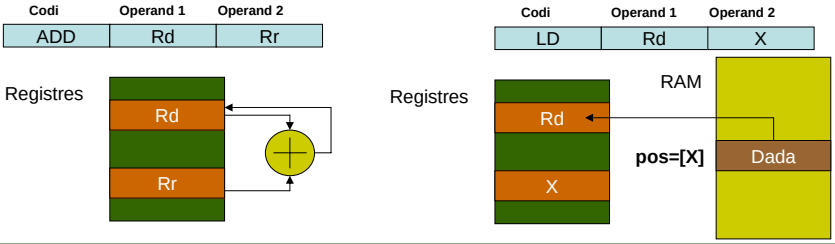
Instruccions: repertori

- El processador és capaç d'interpretar i executar un programa escrit en llenguatge màquina. Al conjunt d'instruccions màquina se l'anomena repertori d'instruccions.
- El repertori d'instruccions d'un processador ha de ser complert i eficaç. Això vol dir que ha de ser capaç de realitzar tot càlcul en un temps finit i emprant un conjunt de recursos no excessiu.
- Per tant, el joc d'instruccions pot ser realment simple. Per exemple, la màquina de Turing empra només 4 instruccions: escriure, moure una posició a l'esquerra i llegir, moure una posició a la dreta i llegir, i finalment parar.
- En general, un processador disposa d'un conjunt una mica més ampla d'instruccions, que es poden classificar en:

Tipus instrucció	Exemple	
Aritmètiques i lògiques	ADD Rd, Rr ANDI Rd, ct	; Sumar el contingut de dos registres: Rd ← Rd + Rr ; AND amb registre i constant: Rd ← Rd AND ct
De transferència de dades	MOV Rd, Rr LD Rd, X	; Transferència entre registres Rd ← Rr ; Adreçament indirecte: Rd ← (X)
De comparació	CP Rd, Rr	; Comparar dos registres: Rd – Rr → activa flags
De salt, condicional i incondicional	BREQ pos JMP pos	; Salt condicional: if (Z=1) then PC ← PC + pos + 1 ; Salt incondicional: PC ← pos
De salt a subrutina	CALL k	; Crida a subrutina que es troba en posició k
Tractament bit a bit i altres	SBI A, b	; Posar el bit b del registre A a 1-lògic
D'entrada / sortida	IN Rd, PIND	; Escriure en registre Rd el valor del port d'entrada PIND
Altres	NOP	; No fer res

Instruccions: format

- El format és la representació de la instrucció
- Especifica el significat de cada bit que composa la instrucció
- Una instrucció ha de contenir la següent informació
  - El codi d'operació. Especifica l'operació a realitzar
  - L'adreça dels operands
  - L'adreça del resultat
  - L'adreça de la propera instrucció
  - El tipus de representació dels operands
- Exemples:



Instruccions: modes d'adreçament (I)

- El mode d'adreçament és el procediment pel que es determina un operand, o la ubicació de l'operand o una instrucció
- Una classificació genèrica dels adreçaments inclou els adreçaments (la major part d'ells amb múltiples variants) és:
  - Immediat
  - implícit
  - Directe
    - Absolut de registre, de memòria o d'índex
    - Relatiu respecte al comptador de programes, respecte a registre, respecte a registre índex o respecte a pila
  - Indirecte

- Adreçament immediat
  - Quan l'objecte (aquí passa a ser l'operand) es troba en la pròpia instrucció
  - No calen referències a memòria

  Exemple: LDI rd, 0xFF ;Es carrega en regsitre rd el valor =xFF

- Implícit
  - Algunes instruccions suposen la ubicació de l'operand, pel que no es troba explícit en la instrucció.

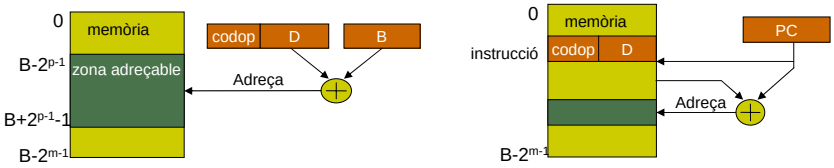
  Exemple: CLC ; Clear carry

Instruccions: modes d'adreçament (II)

- Directe (absolut)
  - En la instrucció s'expressa l'adreça real de l'objecte.
  - Per tant, la instrucció té l'adreça efectiva de la dada

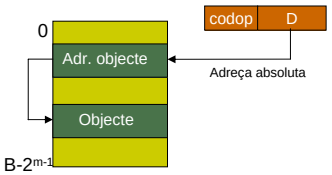
  Exemple: LDS, Rd, pos; Rd ← (pos) ; Càrrega directa de RAM

- Directe relatiu
  - La instrucció té un desplaçament sobre l'adreça marcada per un punter.
  - Per calcular l'adreça efectiva cal sumar (positiu o negatiu) aquest desplaçament a l'apuntador
  - Té l'avantatge de poder canviar l'adreça de treball actuant sobre el valor que pot estar emmagatzemat en un registre
  - Té les variants de poder ser executat sobre un registre base o sobre el comptador de programes
  - Quan es fa sobre un registre índex (variant que incrementa a cada pas el registre índex) permet recórrer de forma fàcil una taula que comença sobre una adreça base.



Instruccions: modes d'adreçament (III)

- Adreçament a pila
  - És una variant del cas anterior, molt emprat en microprocessadors i microcontroladors
  - El processador disposa d'un registre anomenat stack que adreça una posició de la memòria principal (on s'hi posa la pila) que actua com a capçalera de pila
  - Quan es guarda una dada en la pila, el stack s'incrementa (o decrementa, segons on creixi la pila) per apuntar a l'adreça següent per quan arribi una nova dada
  - Si, al contrari, es treu una dada de la pila, amb pre-decrement (o pre-increment) del stack s'adreça a la darrera dada guardada
  - La pila,apart de poder ser accedida per instruccions concretes de treball sobre ella (PUSH, POP), és molt útil quan es treballa amb subrutines la que permet guardar automàticament l'adreça de retorn
- Indirecte
  - Comença com un adreçament directe. L'objecte apuntat, conté l'adreça de l'objecte que es busca.
  - És un doble adreçament
  - La indirecció pot ser afegida a tots els adreçaments vistos
  - És útil per accedir a taules d'apuntadors.



El microprocessador

- **Inicialment un processador estava format per un conjunt discret de sistemes. En els anys 70, quan un sol xip ho va integrar tot, es va començar a parlar de microprocessadors.**
- **La caracterització d'un microprocessador ve determinada per:**
  - El repertori d'instruccions que pot executar
    - Inclou instruccions aritmètiques (sumar, restar, ...), lògiques, comparacions entre dades, salts de programa, crida a subrutines, etc.
  - El tipus d'adreçament
    - La forma com es pot accedir a les dades emmagatzemades en memòria.
  - El nombre de bits per instrucció
    - És el nombre de bits que es tracten per instrucció. El primer microprocessador (l'Intel 4004) treballava amb dades de 4 bits. Això vol dir que la operació suma es restringia a sumes de 4 bits amb 4 bits. Actualment es treballa amb processadors amb amples de bus de 32 bits i 64 bits.
  - L'arquitectura del processador
    - Estableix el mecanisme d'accés que hi ha entre processador i dades i programa:Von Neumann i Harward,etc
  - La complexitat de les instruccions
    - Els processadors RISC (Reduced Instruction Set Computer) té un conjunt reduït d'instruccions màquina, el que implica arquitectures simples i, de retruc, en processadors més barats.
    - Els processadors CISC (Complex Instruction set Computer) tenen un bon repertori d'instruccions i poden executar instruccions màquina més complexes en un mateix cicle de rellotge. Per exemple, l'arquitectura PC d'IBM es basava en aquest tipus de processador. En contrapartida, i degut a aquesta major complexitat, és més difícil implementar en ells estructures de paral·lelisme o de pipelining.
  - La velocitat del rellotge
    - En determina el ritme de treball del processador. A major freqüència, (en principi) major nombre d'instruccions per segon.

Microcontrolador vs microprocessador

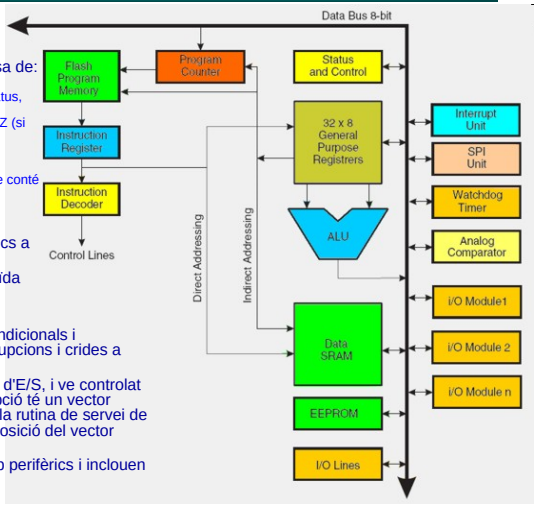
- Els microprocessadors formen la part central de procés en la construcció de computadors o sistemes computacionals grans. Això és, per a comunicar-se amb l'exterior s'han d'acoblar a mòduls perifèrics mitjançant diversos busos de sistema.
- El microcontrolador està format per un microprocessador i el conjunt de subsistemes que normalment té un microprocessador, memòria i entrada/sortida, tot integrat en el mateix circuit.
  - En la construcció de sistemes computacionals específics el cost del sistema es redueix quan s'empren microcontroladors enfront a microprocessadors
  - Els microprocessadors solen basar-se en una arquitectura Von Neuman, a diferència de la Harvard que empren els microcontroladors
  - Els microcontroladors tenen una disposició de registres preparada per a entrada/sortida ràpida, mentre que els microprocessadors treballen més amb acumuladors com a emmagatzemadors per a resultats intermitjos

Microcontroladors ATmega

- Els microcontroladors AVR (ATMEL) són una família de microcontroladors amb diverses subfamílies de microcontroladors amb ampla de bus de 8, 16 i 32 bits que ofereixen un espectre diversificat de possibilitats d'ús en múltiples aplicacions.
- Empren una arquitectura Harward amb memòries i busos separats per programes i dades. Els microcontroladors inclouen un pipelining de fins a tres etapes, amb pre-fetch d'instrucció, que permet aproximar el rendiment de la CPU a 1 instrucció per cicle de rellotge.
- Tenen fins a 256K Bytes de Flash, 4K Bytes de EEPROM i 8K Bytes SRAM. Els programes s'emmagatzemen en la flash.
- El cicle de treball de la CPU és d'1 cicle en la major part d'instruccions, i pot treballar a freqüències de rellotge de fins a 20Hz/segon.
- Tots els microcontroladors consten de la CPU i de perifèrics diversos, com ara ports d'E/S, comptadors/timers, mòduls d'entrada analògics (comparadors/ADCs), ports de comunicació i temporitzadors.
- Els microcontroladors de la mateixa família difereixen en la memòria de programa i el nombre de perifèrics de què es composa cada microcontrolador. Tots els microcontroladors de la família comparteixen el mateix repertori d'instruccions i mateixos adreçaments. Això permet triar el microcontrolador més adient per a cada aplicació.

ATmega8: CPU

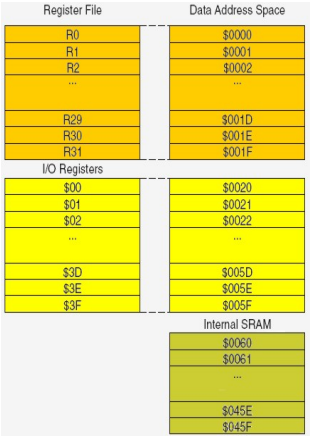
- El microcontrolador ATmega8 és un dels més simples de la família.
- La CPU dels microcontroladors de la família ATmega disposa de:
  - ALU, amb operacions aritmètico-lògiques entre operands emmagatzemats en registres o amb immediats. El registre de status, s'actualitza a cada operació de la UAL, conté informació sobre el resultat obtingut i guarda flags de control d'operació com els bits Z (si zero), C (si carry) i N (si negatiu)
  - 32 registres de 8 bits de propòsit general que permeten realitzar operacions aritmètiques en 1 cicle de rellotge.
  - Comptador de programes, que apunta a l'adreça de memòria que conté la nova instrucció
  - Registre d'instruccions, que conté la instrucció que s'executa
  - Decodificador d'instruccions
- La CPU es comunica amb la memòria de dades i els perifèrics a través del bus de sistema de 8 bits
- El programa s'emmagatzema en una memòria flash construïda sobre el mateix microcontrolador.



- El flux del programa es controla amb instruccions de salt condicionals i incondicionals que adrecen tot l'espai de memòria. En interrupcions i crides a subrutines l'adreça de retorn s'emmagatzema en el stack.
- El mòdul d'interrupcions té els registres de control en l'espai d'E/S, i ve controlat per un bit de capacitació global d'interrupcions. Tota interrupció té un vector d'interrupció en la taula de vectors d'interrupció per adreçar la rutina de servei de la interrupció. La prioritat de la interrupció s'estableix en la posició del vector d¡interrupció dins la taula de vectors.
- L'espai de memòria d'E/S conté 64 adreces per accions amb perifèrics i inclouen registres de control, comunicació i E/S.

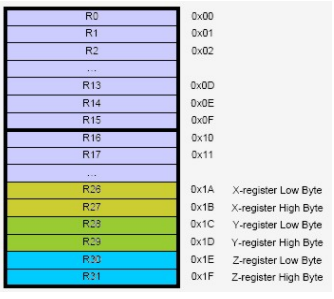
ATmega8: mapa de memòria (I)

- L'arquitectura AVR té dos espais de memòria, el de programa i el de dades. També conté una memòria EEPROM addicional per dades.
- Memòria flash
  - El programa es carrega en una memòria flash de 8 KBytes, organitzada com a 4K x 16 bits (les instruccions del AVR són de 16 o 32 bits). El comptador de programes és de 12 bits i adreça tot l'espai de memòria (les 4K adreces de memòria).
  - El programa pot contenir taules de constants que són emmagatzemades en la flash emprant la instrucció LPM (*Load Program Memory*).
- Memòria SRAM
  - L'espai de memòria SRAM global inclou 1120 bytes, i es composa de l'espai de memòria d'E/S (que correspon a les 96 primeres adreces) i a la memòria interna SRAM (les 1024 posicions restants).
  - La SRAM es fa servir per guardar dades en temps d'execució
  - L'espai de memòria té 5 modes d'adreçament: directe, indirecte, indirecte amb desplaçament, indirecte amb pre-increment i indirecte amb post-increment.
  - L'adreçament directe es pot fer en tot l'espai de memòria.
  - Els registres apuntadors en el mode indirecte són els X, Y i Z.
    - L'adreçament indirecte amb desplaçament només admet un desplaçament de 63 posicions des de l'adreça base i es fa amb els registres Y o Z.
    - El desplaçament indirecte amb pre-increment o post-increment es fa amb els registres X, Y o Z.



ATmega8: mapa de memòria (II)

- L'espai de memòria d'E/S
  - L'espai de memòria d'E/S de l'ATmega8 el formen els registres d'E/S i els regsitres de status i de control dels direrents perifèrics que configuren tot el microcontrolador.
  - La particularitat d'aquest espai de memòria es troba en les instruccions que es poden emprar per treballar-hi:
    - **Instruccions IN i OUT**. L'espai de memòria d'E/S pot ser accedit emprant les instruccions IN i OUT, transferint dades entre els 32 registres de propòsit general i l'espai d'E/S. El rang d'adreces va de 0x00 a 0x3F.
    - **Instruccions SBI, CBI, SBIS i SBIC**. Els registres d'E/S que es troben en el rang d'adreces 0x00 a 0x1F poden emprar aquestes instruccions que faciliten l'accés o permeten preguntat per l'estat en bits concrets.
    - **Instruccions LD i ST**. Quan els registres d'E/S es facin servir com a adreçament en l'espai de dades mitjançant aquestes instruccions s'ha d'afegir 0x20 a aquestes adreces.
- El fitxer de registres
  - Està constituït per 32 registres de 8 bits, mapejats en l'espai de memòria de dades: de l'adreça 0x00 a la 0x1F.
  - Hi ha 6 registres que, emprats dos a dos, fan d'apuntadors de 16 bits en l'adreçament indirecte. S'anomenen X, Y i Z, i agrupen els registres R27-R26, R29-R28, R31-R30, respectivament.
  - El registre Z també fa apuntador d'adreces en taules de cerca (*look-up table*) en la memòria flash per programa.
- El stack
  - El *stack* es posa efectivament en la memòria SRAM. S'ha d'inicialitzar en cada programa prèviament a ser utilitzat.
  - El *stack pointer* és l'apuntador del *stack* que referència l'adreça a escriure o llegir quan s'accedeix al *stack*. Inicialitzar a la posició més alta de la SRAM, el *stack pointer* decrementa en operacions de posar dades (*push*) i s'incrementa en treure-les (*pop*) .



ATmega8: repertori d'instruccions (I)

- El repertori d'instruccions conté instruccions aritmètico-lògiques, instruccions de càrrega i d'emmagatzemament, instruccions de salt i és potent en instruccions d'entrada/sortida, contenint instruccions que permeten treballar individualment amb cada pin.
- El repertori d'instruccions del ATmega8 el trobeu en el corresponent full d'especificacions del component:

  http://www.atmel.com/dyn/resources/prod_documents/doc2486.pdf

- Algunes consideracions a tenir en compte són:
  - Les operacions aritmètiques poden emprar els registres R0 a R31, però no es fan sobre RAM. Pràcticament totes es fan en un cicle excepte les sumes amb paraules (16 bits) i la multiplicació.
  - A la RAM i a l'espai d'entrada/sortida només s'hi pot accedir via registres o per còpia de dada. L'accés indirecte es fa amb els registres X, Y i Z.
  - Els accessos a la RAM tarden 2 cicles, i la lectura de la RAM amb LPM en tarda 3.
  - Hi ha dos tipus de salts condicionals
    - Els salts (BRxx). Sobre comprovació de flag poden saltar a una adreça especificada
    - Els skips (SBxx). Sobre comprovació de bit arbitrari (en registre d'E/S) poden saltar a la propera instrucció si es compleix la condició

ATmega8: repertori d'instruccions (II)

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← 0xFF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← 0xFF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2

ATmega8: repertori d'instruccions (III)

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1 / 2 / 3
CP	Rd,Rr	Compare	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd – K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1 / 2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1 / 2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1 / 2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1 / 2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1 / 2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1 / 2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1 / 2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1 / 2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1 / 2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1 / 2

ATmega8: repertori d'instruccions (IV)

Mnemonics	Operands	Description	Operation	Flags	#Clocks
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2

ATmega8: repertori d'instruccions (V)

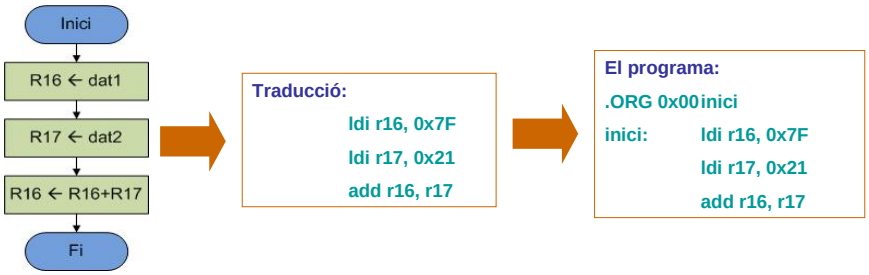
Mnemonics	Operands	Description	Operation	Flags	#Clocks
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1

Assemblador: introducció

- L'assemblador és un llenguatge de baix nivell. per tant, les instruccions són simples i directament relacionades amb el maquinari.
- Com s'ha vist en les pàgines anteriors, cada instrucció relaciona components o perifèrics específics de la màquina o permet obtenir informació (preguntant sobre l'estat del registre status) sobre el procés acabat d'executar, el que ens permet saltar el seqüenciament previst pel comptador de programes.
- L'assemblador és molt sensible als errors. Empreu sempre un simulador (AVRstudio, per exemple) per a comprovar el funcionament dels programes
- La pràctica en programació exigeix la realització de diagrames de flux com a medi d'especificació i clarificació en el disseny de programes
- És altament recomanable comentar adequadament el codi que es genera. Els comentaris poden ser en tres nivells:
  - A començament de programa. Escriure el nom del programa, l'autor, la data, la revisió i una descripció del què fa el programa
  - En cada inici de mòdul. Per marcar clarament un nou procediment, funció o subrutina, indicant-ne el nom i els recursos emprats
  - En cada línia per aclarir detalls particulars

Assemblador: elPrimerPrograma.v1

- El primer programa, molt simple:
  Sumar dues dades immediates de dos registres i guardar el resultat en el primer registre



Quin és el status de la CPU i què hi ha carregat en memòria (flash, SRAM) ?

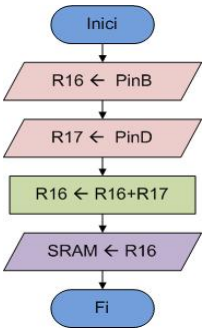
... i quan s'acaba d'executar la suma?

...quin valor han pres els registres r16 i r17?
...per què no s'ha escrit res en la SRAM?

(nota: en el mapa de memòria I/O hi ha canvis en les posicions 11 i 45, però són deguts a inicialitzacions de registres d'E/S que encara no s'han vist)

Assemblador: elPrimerPrograma.v2

- El programa anterior només suma dues dades que a més es carreguen de forma immediata en dos registres. De poca cosa serveix!
- Anem a millorar el primer programa:
  S'agafen dues dades entrades pels ports A i B, es sumen, i es guarden en memòria.



Traducció:

```
in r16, PinB
in r17, PinD
add r16, r17
st X, r16
```

...on comença X?

El programa:

```
.INCLUDE "m8def.inc"

.ORG 0x0  rjmp inici

inici: ldi XL, 0x60      ; low(0x60)
       ldi XH, 0x00      ; high(0x60)
       in r16, PINB      ; entrar dat1
       in r17, PIND      ; entrar dat2
       add r16, r17      ; sumar
       st X, r16         ; guardar
```

Si PinA = 0x66 i PinB = 0x22,

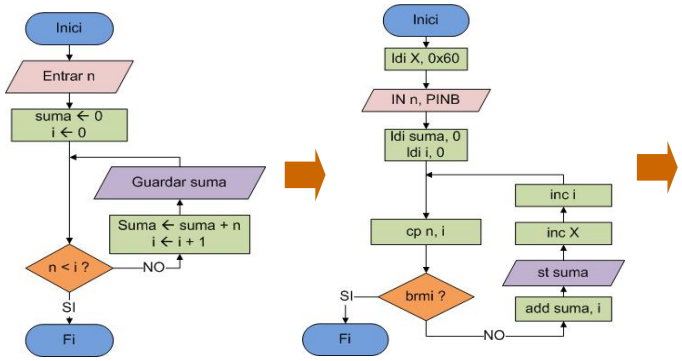
...què valdrà la suma?

...a quina posició es guardarà la dada en la SRAM?

Assemblador: elSegonPrograma

- Ara es treballa amb salts

  Es demana guardar en memòria la suma dels *n* primers naturals, on *n*



El programa:

```
.INCLUDE "m8def.inc"

.DEF n      = R16
.DEF suma   = R17
.DEF i      = R18

.ORG 0x0    rjmp inici

inici:
      ldi XL, 0x60    ; low(0x60)
      ldi XH, 0x00    ; high(0x60)

      in   n, PINB    ; agafar n
      ldi suma, 0     ; suma <-- 0
      ldi i, 0        ; i <-- 0
mes:cp n,i
      brmi fi
      add suma, i     ; sumar
      st   X+, suma   ; guardar
      inc i
      rjmp mes
fi:   rjmp fi
```

Memòries i registres X, Y i Z (I)

- En l'exercici anterior s'ha treballat amb el registre índex X.
- Recordar que X, Y i Z actuen com a índexs de 16 bits i corresponen a les concatenacions dels registres r26:r27, r28:r29 i r30:r31, respectivament
- Aquests registres s'empren en l'adreçament a memòria, però ...amb detalls
- **Adreçaments**
  - **Directe**: s'emmagatzema a una adreça de memòria concreta un valor
    Exemple:

```
ldi r16, 55        ; valor a guardar
sts 0x70, r16      ; es guarda temps a l'adreça 0x70
```

  - **Indirecte**: s'empren els apuntadors X, Y, Z (Y i Z admeten, a més, desplaçament)
    Exemple (com en l'exercici anterior):

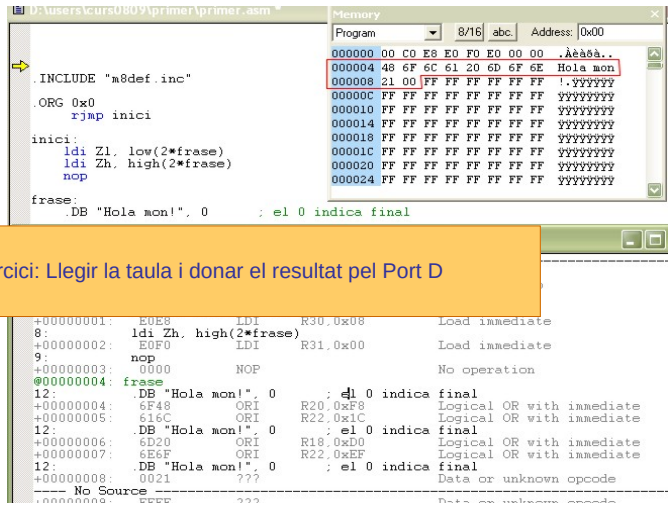
```
ldi r16, 0x44      ; valor a guardar
ldi Xh, 0x00       ; adreça X, byte alt → r27
ldi Xl, 0x60       ; adreça X, byte baix → r26
st X, r16          ; es guarda temps a l'adreça X
inc Xl             ; s'incrementa X (compta que s'ha de fer servir el byte baix!)
st X, r16          ; es guarda temps a l'adreça X+1
```

  - Compta amb l'espai de memòria:
    - Les adreces de la RAM que comprenen l'espai d'entrada/sortida (entre 0x20 i 0x5F) poden ser adreçades directament emprant IN / OUT.
    - Però també ho poden ser emprant LDS i STS. En aquest cas, cal sumar 0x20 a l'adreça, ja que l'espai de memòria d'E/S, que va de 0x00 a 0x3F està mapejat en l'espai global amb les adreces 0x20 a 0x5F (veure el mapa de memòria).

Memòria de programa: tercer programa

- **...i la flash?**
  - Organitzada en paraules de 2 bytes → A tenir en compte quan s'hi treballi...
    - Quan es guarden dades sempre es guardaran per paraules (2 bytes)
    - Quan s'adreci, l'adreça física és la doble de l'apuntador, ja (
      paraula són 2 b
  - L'apuntador Z per
    treballar amb la fl

    Exemple: Guardant una taula



Exercici: Llegir la taula i donar el resultat pel Port D

Exercicis

- Sèrie de Fibonacci
  - La sèrie de Fibonacci es construeix a partir de la suma dels dos nombres anteriors ($x_n = x_{n-1} + x_{n-2}$): 1, 1, 2, 3, ...
  - Partint de la llavor (els dos primers nombres) emmagatzemar en memòria els 13 primers termes de la sèrie
- Càlcul del sinus
  - El programa trobarà el Sinus d'un angle donat.
  - L'angle s'entrarà pel PortB, i el Sinus es donarà pel PortD (multiplicat per 100)
  - El càlcul es realitzarà a partir d'una taula emmagatzemada en la flash
    - Nota: La taula emmagatzemada contemplarà els angles 0 a 90, en passos de 10 (es deixa llibertat per a crear una taula més precisa!)

	0	10	20	30	40	50	60	70	80	90
Sinus	0	17	34	50	64	77	87	94	98	100

Entregar els dos exercicis en una memòria en la que, per a cada exercici, hi hagi...

...el diagrama de flux a alt nivell

...si cal, el diagrama de flux a baix nivell

...el programa

...i imatges (impressions de pantalla) on es vegi el correcte funcionament del programa

Bibliografia

- Pedro de Miguel Anasagasti. Fundamentos de los Computadores. Edit Thomson. 2004.
- ATmega8 Datasheet
- http://www.cannic.uab.es/Docencia/FCwebAVR/FC_AVR.htm