**Programació de Microprocessadors**

Miquel Rodríguez Juvany

2on Trimestre

**Sessió 3: Introducció (Microprocessador – Microcontrolador)**

**Número d’instruccions**

MPU: Màquina que ha de poder fer moltes coses

MCU: Orientació molt específica

**Tipus d’instruccions**

MPU: Habitualment té instruccions de tot tipus:

* Simples
* Complexes
* Grans
* Petites
* Ràpides
* Lentes

MCU: Té instruccions molt uniformes

|  |  |  |
| --- | --- | --- |
|  | **MPU** | **MCU** |
| **Número d’instruccions** | Màquina que ha de fer moltes coses | Orientació molt específica |
| **Tipus d’instruccions** | Instruccions de tot tipus: simples, complexes, grans, petites, ràpides, lentes... | Instruccions molt uniformes |
| **Tipus d’aplicacions** | Utilitzat en entorns de alta demanda de capacitat de procés | Utilitzat per tasques molt específiques (concretes): unes poques |
| **Mida** | Més gran | Mínima possible |
| **Capacitat I/O (Analògiques / Digitals)** | Cap directament.  Com es comunica llavors? | I/O Analògiques.  I/O Digitals Integrades. |
| **Connectors perifèrics** | USB  Ethernet  Wi-Fi | UART  I2C  SPI |
| **Mode de funcionament** | Pensat per entorns amb Sistema Operatiu potent.  Tasques diverses.  Tasques inclús concurrents. | Pensat per entorns repetitius.  Màquina d’estats.  Un cop posat en marxa no és fàcil canviar la tasca |

**Comunicacions sèrie-paral·lel**

Gráfico, Diagrama

Descripción generada automáticamenteDiagrama, Esquemático

Descripción generada automáticamente

Perquè internament tenim comunicació paral·lela i externament sèrie?

Perquè amb la comunicació paral·lela és més probable que el “soroll” d’una senyal “molesti” a una altra senyal i l’altre dispositiu funcioni pitjor, amb això internament no hi ha problema però, externament, els perifèrics perden eficiència.

Què em dona la comunicació sèrie?

La possibilitat de tenir molts perifèrics sense interferències.

Diagrama

Descripción generada automáticamente

**Comunicació sèrie síncrona-asíncrona**

Síncrona:

* La comunicació és en forma de blocs (frames)
* Més ràpida
* Més senzilla
* Interval constant per la transmissió
* Requereix de mecanisme de rellotge (precís)

Asíncrona:

* La comunicació és en forma de caràcters/bytes
* Més lenta
* Menys senzilla
* Interval de transmissió no constant. Pausa entre bit de parada-inici (menys eficiència de canal).

**Comunicacions sèrie-paral·lel**

Diagrama, Texto

Descripción generada automáticamente con confianza media

Escala de tiempo

Descripción generada automáticamente

**UART**

*Universal Asynchronous Reciever/Transmitter*

Transforma comunicacions paral·leles en comunicacions sèrie.

**USART**, també té la possibilitat de comunicació asíncrona.

Algunes de les funcions de la UART en transmissió:

* Serialitzar les dades (A = 65 = 0100 0001)
* Invertir-les (enviem 1000 0010 🡪 0100 1101)
* Bit de inici / parada.
* No té rellotge. Sinó que la comunicació ve marcada pels bits d’inici i parada.

Comunicació **entre dos**, no permet més dispositius.

No és bidireccional.

Distàncies curtes.

Gráfico, Diagrama

Descripción generada automáticamenteDiagrama, Esquemático

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Avantatges:

* Treballa amb 2 línies
* No requereix de rellotge
* Control de paritat
* Canvi dinàmic de la estructura de paquet de dades (només cal que els dos extrems ho acordin) (marcat per bit inici/parada).

Inconvenients:

* La mida de la trama de dades màxim 9 bits
* No admet ni múltiples slaves, ni múltiples masters (és comunicación un-a-un).
* Comunicació half-dúplex (no és full-dúplex)

1 🡪 UART-Transmissora rep les dades del bus (en paral·lel)

2 🡪 UART-Transmissora serialitza les dades i afegeix: bits d’inici, paritat i parada.

3 🡪 UART-Transmissora envia el paquet/frame a través de la connexió sèrie asíncrona cap a UART-Receptora, que anirà mostrejant la línia de dades a la taxa marcada. (TX 🡪 RX)

4 🡪 UART-Receptora descarta (després d’utilitzar-los) els bits d’inici, paritat i parada del frame rebut, i prepara les dades per enviar-les pel bus de dades.

5 🡪 UART-Receptora envia les dades pel bus paral·lel.

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza baja

**SPI**

*Serial Peripheral Interface*

Dissenyat per enviar un número variable de bits, sense interrupció.

Aquest conjunt de bits s’anomena packet.

Comunicació **master-slave**:

* Master acostuma a ser el MCU
* Slave: Acostuma a ser un sensor, display, memòria...
* Relació: **1:n**

Bus de comunicació format per:

* MOSI (*Master Output/Slave Input*): enviament de dades de master a slave.
* MISO (*Master Input/Slave Output*): enviament de dades de slave a master.
* SCLK (*clock*): pel senyal de rellotge.
* SS/CS (*Slave Select/Chip Select*): perquè el master seleccioni el slave amb el que es vol comunicar.

Sistema de comunicació master-slave:

* El clock generat i configurat a nivell de master
* Com té rellotge és una comunicació asíncrona
* Seleccionem el slave per activació de la línia SS/CS (una per slave o per daisy-chain)

Avantatges:

* No treballa amb bits inici/parada.
* Per tant: flux continu de bits.
* Adreçament de slave senzill.
* Més velocitat de transmissió que I2C (doble aprox.)
* Bidireccional (MOSI/MISO) (full dúplex)

Inconvenients:

* Treballa amb 4 línies
* No té confirmació que les dades arriben bé
* No té control de paritat
* Només admet 1 master

Gráfico

Descripción generada automáticamente

**I2C**

*Inter-Integrated Circuit*

Bus de comunicació format per:

* SDA (Serial Data) (han de tenir resistència de pull-up)
  + Línia per enviament/recepció de dades
* SCL (Serial Clock) (han de tenir resistència de pull-up)
  + Línia per enviar clock

Sistema de comunicació master-slave

* Master genera senyal de clock per SCL per sincronitzar dos extrems de la comunicació
* Cada slave té una adreça de 8 bits (7 identificació, 1W/R)
* Comunicació:
  + Tots els slaves tenen adreça
  + Master envia adreça de slave+flag R/W
  + Slave es prepara per R/W
  + S’efectua la comunicació entre slave/master
  + Es torna a alliberar canal/bus

Avantatges:

* Només 2 línies
* Permet múltiples masters/slaves
* ACK/NACK per cada trama
* HW més senzill que UART

Inconvenients:

* Taxa de transferència menor que SPI
* Mida de data frame limitada a 8 bits
* HW més complicat que SPI
* Comunicació half-dúplex

Diagrama

Descripción generada automáticamente

**Bit Inici** 🡪 Indica comencem

**Adreça trama** 🡪 7/10 bits que identifica cada slave

**Flag R/W** 🡪 Low

**Emissions electromagnètiques**

Què són? Com afecten al dispositiu? Què les hem de considerar d’entrada? O de sortida? Les consideracions entre MCU-MPU han de ser diferents? Tenim requeriments diferents?

**Rang temperatura de treball**

Què són? Com afecten al dispositiu? Les consideracions entre MCU-MPU han de ser diferents? Tenim requeriments diferents?

**Velocitat d’engegada**

En quina forma ens afecta? Els requeriments entre MCU-MPU han de ser diferents?

**Mode de funcionament**

MPU:

* Pensat per entorns amb S.O. potent
* Tasques diverses
* Tasques inclús concurrents

MCU:

* Pensat per entorns repetitius
* Màquina d’estats
  + Un cop posat en marxa... no és fàcil canviar la tasca

Imagen que contiene Texto

Descripción generada automáticamente

**Sessió 4: Introducció (Microprocessador – Microcontrolador)**

Diagrama

Descripción generada automáticamente**Màquina d’estat**

Funcionament en mode cíclic: ARDUINO

* Repetició continuada de les mateixes accions
* Un únic programa
* Que no acaba mai

**Arquitectura RISC / CISC**

CISC: **Complex Instruction Set Computer**

* Conjunt gran d’instruccions
* Instruccions poden ser complexes o simples
* Instruccions poden ser llargues o curtes
* Instruccions poden trigar molt o poc en executar-se

RISC: **Reduced Instruction Set Computer**

* Conjunt petit d’instruccions
* Instruccions “totes” de la mateixa complexitat (simples)
* Instruccions “totes” de la mateixa llargària (curtes)
* Instruccions “totes” amb el mateix temps d’execució (1 cicle de rellotge)

**Arquitectura 1 bus / 2 busos**

Arquitectura d’un bus:

* Von Neumann / Princeton
* La mida de l’únic bus marca la mida de totes les informacions a intercanviar entre CPU / unitats
  + Comunicació més lenta

Arquitectura de dos busos:

* Harvard
* Dos busos: un per dades, un per instruccions
  + Comunicació més ràpida
  + PCB més complicat
  + Memòria de dades / instruccions poden ser diferents

Interfaz de usuario gráfica, Texto, Aplicación, Tabla

Descripción generada automáticamente**Endianness**

MSB: Most Significant Byte

LSB: Less Significant Byte

**Little Endian**: DEC-Intel

* Ordenem la informació de LSB 🡪 MSB

**Big Endian**: RISC-Motorola-TCP/IP (Network Order)

* Ordenem la informació de MSB 🡪 LSB

**MIPS / MFLOPS**

Com podem mesurar les prestacions d’un MICRO:

* **MIPS**: Millions of Instructions per Second
  + Fa referència a instruccions
* **MFLOPS**: Millions of Floating Point Operations per Second
  + Fa referència a operacions

**Sessió 5**

**Introducció: Procés execució programes**

Com funciona l’execució de programes?

1. Agafar instrucció
2. Interpretar-la
3. Anar a buscar operació
4. Fer operació
5. Guardar resultats

* STEP 1: Agafar la instrucció a executar
* STEP 2: descodificar la instrucció (interpretar-la)
* STEP 3: executar la instrucció
* STEP 4: escriure el resultat de la instrucció (en memòria)



**Decodificació: Codi operació + Operands extrets de la instrucció**

Diagrama

Descripción generada automáticamente

**Procés execució programes**

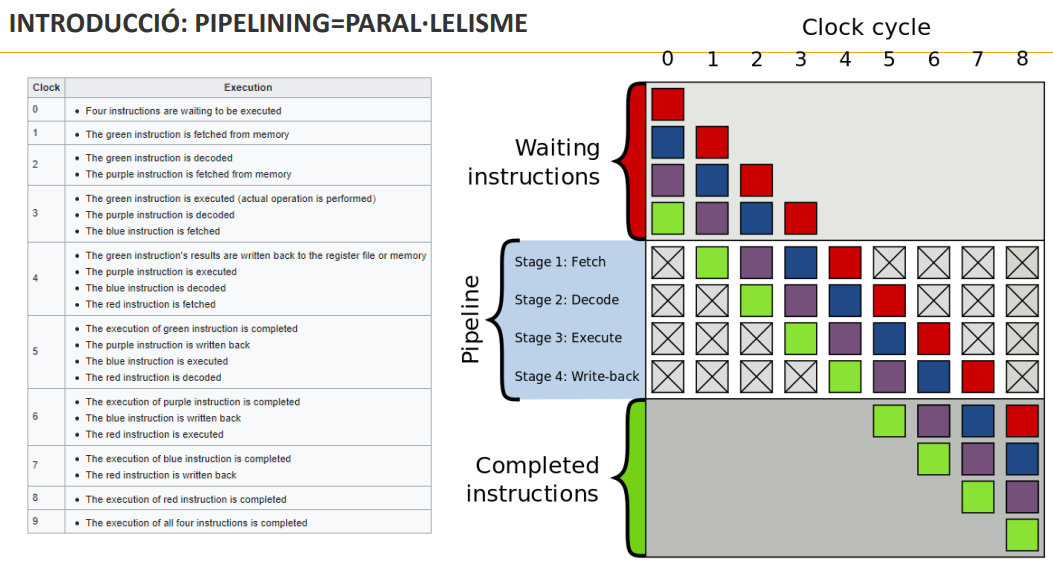
Si fem execució seqüencial:

F1-D1-X1-S1-F2-D2-X2-S2...

Com cada instrucció té 4 fases (FETCH DECODE EXECUTE STORE)

1. Cada instrucció triga 4 CLK en finalitzar
2. Per tant si tenim 1MHz de velocitat de rellotge
   1. Tindrem 1.000.000 de CLK per segon
   2. Però farem una instrucció cada 4 CLK
   3. Per tant, NOMÉS podrem fer 250.000 instruccions per segon

**PIPELINING = PARAL·LELISME**



**Pipelining**

Diagrama

Descripción generada automáticamente

Amb el Pipelining, tindré un mòdul del micro independent per fer cada sub-operació:

1. FETCH: un mòdul per llegir la instrucció.
2. DECODE: un mòdul per descodificar la instrucció.
3. EXECUTE: un mòdul per executar la instrucció (ALU).
4. STORE: un mòdul per guardar el resultat.

Com en una cadena de producció, en tenir 4 mòduls, independents... Podrem fer les 4 suboperacions de forma paral·lela amb instruccions diferents.

Tabla

Descripción generada automáticamente

Si ara augmentem el número de instruccions del programa, augmentarà el número de cicles per executar-lo, en la mateixa mesura:

* 7 instruccions 🡪 CICLES = 4+3+3 = 10 CLK
* 8 instruccions 🡪 CICLES = 4+4+3 = 11 CLK
* 9 instruccions 🡪 CICLES = 4+5+3 = 12 CLK
* 10 instruccions 🡪 CICLES = 4+6+3 = 13 CLK
* n instruccions 🡪 CICLES = 4 + (n-4) + 3 = n+3 CLK

**Si tenim una MCU**, quantes instruccions executarà en el seu programa?

Com que és una màquina d’estats, només executa un programa per **sempre**. Per tant, executa un número **infinit** de instruccions. Per tant, si n és infinit, podrà executar n instruccions en n+3 cicles però si n tendeix a infinit, n+3 tendeix a n.

Per tant, **podem fer n instruccions en n cicles.**

**Pipelining: Independència d’execució: 1 o 2 busos? RISC / CISC?**

Diagrama

Descripción generada automáticamente

Si tenim 1 bus, no podrem paral·lelitzar perquè no podem fer FETCH a la vegada que STORE. **Ens interessa tenir 2 busos**.

Ens interessa que sigui RISC per poder paral·lelitzar.

**Pipelining: Instruccions llargues. Més d’un cicle de FETCH**

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

**Pipelining: Difícil implementar en CISC**

CISC:

* Instruccions molt diferents
* Molt difícil d’implementar pipelining
* Cicles d’instruccions ràpids: freqüència alta.

**Pipelining: Superescalaritat**

CISC:

* Disposar de vàries unitats per fer suboperacions
  + FETCH
  + DECODE
  + EXECUTE
  + STORE

Interfaz de usuario gráfica, Aplicación, Tabla

Descripción generada automáticamente

**Superescalabilitat**

La superescalabilitat és la capacitat d’un processador de accelerar el pipelining. Disposa de més d’una unitat per alguna o totes les fases (F-D-X-W)

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

**Dependència**

Limitació de la capacitat de paral·lelisme del processador superescalar:

* **Dependència estructural**
* **Dependència de dades**
* **Dependència d’escriptura**
* **Dependència de control**

Les dependències poden detectar-se/resoldre’s de forma estàtica o dinàmica:

* **ESTÀTICA**: abans de la execució del codi (en temps de compilació)
* **DINÀMICA**: a mida que el codi s’està executant

Si el compilador veu que necessita retrassar quelcom. Pot afegir temps d’espera. Instruccions que no fan res. Per tal que l’execució sigui la adequada. Eficient des del punt de vista de pipelining.

**Dependència estructural**

Les instruccions han esgotat el número de unitats funcionals requerides/disponibles. Només tenim dues unitats de fetching, en necessitem tres.

**Dependència de dades**

Si estem pendents de disposar d’una dada que és resultat de una operació anterior, encara no completada:

Texto, Carta

Descripción generada automáticamente

**Dependència d’escriptura**

Quan volem escriure a la mateixa posició de memòria que una altra operació:

* No és un problema d’accés a la memòria (podríem tenir dos busos d’adreces)
* Sinó d’accedir EXACTAMENT a la mateixa posició

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Dependència de control (condicionals)**

Si encara no podem saber si el resultat de la condició serà un o un altre, i per tant no podem començar a paral·lelitzar:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**Performance Equation (Equació de rendiment)**

Com podem mesurar el rendiment d’un processador?



Mesurem el rendiment com el temps que triga en executar-se un programa. Depèn de tres factors:

* Temps de cicle de cada instrucció (la velocitat de rellotge del sistema)
* Número de cicles de cada instrucció executada
* Número de instruccions del programa a executar

Com són aquests factors en màquina CISC/RISC?



* Texto

  Descripción generada automáticamente: Si el rellotge va més ràpid, millor 🡪 CISC optimitza
* : Si tenim instruccions que ocupen més de un cicle, el factor no està optimitzat 🡪 RISC millor
* : Si el programa té menys instruccions, millor 🡪 CISC optimitzat

**Sessió 6**

**Recordem: ATMEGA328P**

Analitzem les característiques de un MICRO

**ATMEGA328P-DataSheet**

*A* ***data sheet, data-sheet*** *or* ***spec sheet*** *is a document that summarizes the performance and other characteristics of a product, machine, component, material, subsystem or software in a suficient detail that allows a buyer to understand what the product is and a design engineer to understand the role of the component in the overall system. Typically, a datasheet is created by the manufacturer and beguins with an introductory page describing the rest of the document, followed by listings of specific characteristics, with further information on the connectivity of the devices. In cases where there is relevant source code to include. It is usually attached near the end of the document or separated into another file. Datasheets are created, stored, and distributed via product information management, or product data management systems.*

Document de referència del dispositiu. Dades directes del fabricant.

Document a llegir sempre abans de començar a treballar.

Texto

Descripción generada automáticamente

És un **microcontrolador**.

De 8 bits 🡪 **ports de 8 bits**.

Amb memòria FLASH 🡪

* In-System 🡪 **INTERNA MCU**
* Programable
* Mida:
  + 4KB
  + 8KB
  + 16KB
  + 32KB
* DataSheet conjunt de ATmega48PA, ATmega88PA, ATmega168PA, ATmega328P.
* Texto

  Descripción generada automáticamenteMicrocontrolador de 8 bits.
* **Baix consum**
* **Altes prestacions**
* AVR
* Arquitectura **RISC**, Avançada:
  + **131 instruccions** (potents)
    - La majoria de UN CICLE DE RELLOTGE D’EXECUCIÓ
  + **32 x 8 Registres de propòsit general**
  + Operació estàtica 🡪 **Màquina d’Estats**
    - Carregar programa. Arrancar. Executar.
  + Arriba a **20MIPS** amb rellotge de **20MHz**
  + Integrat un multiplicador de 2 cicles (2 clocks)
* Diagrama

  Descripción generada automáticamentePer a què utilitzem els registres:
  + Per tenir els arguments (operands) de les operacions que farà la ALU
* Els registres de propòsit general
  + Serveixen per tot
  + Veurem després registres que de propòsit particular
* ATmega328P
  + En té 32... R0, R1, ..., R30, R31
  + Tots de 8 bits 🡪 Recordem MCU de 8 bits
* Imagen de la pantalla de un celular con texto

  Descripción generada automáticamente con confianza mediaDiversos segments de Memòria:
  + Que emmagatzemen informació durant força temps
  + 1: **Memòria FLASH**, programable, per emmagatzemar codi, integrada en MCU 🡪 **NON-VOLATILE RAM**
    - Funcionament:
      * Podem llegir 1 BYTE
      * Podem escriure (si no hi ha res) 1 BYTE
      * Però, si hem de modificar només ho podem fer per blocs de 128 BYTES.
    - 4KB 🡪 ATmega48PA
    - 8KB 🡪 ATmega88PA
    - 16KB 🡪 ATmega168PA
    - **32KB 🡪** ATmega328P 🡪 podrem tenir codi de fins a 32KB 🡪 (aproximadament) 32K instruccions
  + 2: **Memòria EEPROM**, programable? per emmagatzemar què?
    - Encara que apaguem continua emmagatzemat
    - Com si fos un MINI-disc dur
    - Paràmetres de configuració, que han de perdurar a apagada
      * INFORMACIÓ CRIPTOGRÀFICA
      * INFORMACIÓ DE CLAUS/CONTRASENYES
    - Informació llegida que ha de perdura si s’apaga
      * LECTURA DE SENSORS
    - Funcionament:
      * Podem llegir 1 BYTE
      * Podem escriure 1 BYTE
    - 256B 🡪 ATmega48PA
    - 512B 🡪 ATmega88PA
    - 512B 🡪 ATmega168PA
    - **1KB** 🡪 ATmega328P
  + 3: **Memòria SRAM**: per informació VOLÀTIL, com variables.
    - **Programa**: Perquè en FLASH i no en EEPROM
      * Tecnologia FLASH (NAND) vs EEPROM (NOR)
        + + Velocitat
        + + Densitat
        + – Cost
        + Un cop configurat no es canvia el programa
    - Cicles d’escriptura / lectura:
      * Quantes operacions de lectura / escriptura podem fer
    - **FLASH:** 10.000
    - **EEPROM:** 100.000
    - 512B 🡪 ATmega48PA
    - 1KB 🡪 ATmega88PA
    - 1KB 🡪 ATmega168PA
    - **2KB** 🡪 ATmega328P
  + Retenció de dades en memòria: 20 anys / 100 anys.

**Sessió 7: Estructura ATMega328P**

**Recordem DataSheet**

Perifèrics (I/O):

* 2 comptadors/temporitzadors
  + 8 bits
* 1 comptador/temporitzador
  + 16 bits
* Comptador real time, amb oscil·lador separat (pot anar a qualsevol velocitat de cicle)

Comptador:

* màquina que compta events
  + P.e., vegades que s’obre una porta
  + 8 bits: permet comptar de **0 a 255**

Temporitzador:

* màquina que calcula intervals de temps
* Un interval de temps, es basa en:
  + Temps de cicle
  + Número de cicles
  + Si el temps de cicle és de 1 sec, i vull calcular un interval de 100 sec hauré de calcular 100 temps de cicle
* **Comptador** calcula events **asíncrons**
* **Temporitzador** calcula events **síncrons**

6 canals PWM (Pulser-Width Modulation)

6/8 Canals ADC de 10 bits **(Analog-Digital Converter)**

* 10 bits 🡪 0...1023 nivells

Encapsulament

* **TQFP** (Thin Quad Flat Package)
* **QFN**/MLF (Quad-Flat-No Leads / Micro Lead Frame)
* **PDIP** (Plastic Dual In-Line Package)

**Sessió 8: Estructura ATMega328P**

**Gestió PC / Crides a funció: Stack Pointer (SP)**

Diferents modes de funcionament de pila:

1. Apilem de posicions altes de memòria cap a posicions baixes.
2. Apilem de posicions baixes de memòria cap a posiciones altes.

Imagen que contiene Tabla

Descripción generada automáticamente

1. Si apunta a primera posició lliure de la pila
2. Si apunta a darrera posició ocupada de la pila

Escala de tiempo

Descripción generada automáticamente con confianza media

**Funcionament Stack**

Dues operacions:

PUSH 🡪 Posar informació a Stack / moure SP

PULL 🡪 Recuperar informació de Stack / moure SP

1. Apilem cap a munt... SP apunta primera lliure
   1. Push: Guardem dada... pugem SP ADREÇA 🡪 [SP]... SP = SP - 1
   2. Pull: Baixem SP... recuperem dada SP = SP + 1 ... [SP] 🡪 ADREÇA (SP)
2. Apilem cap a munt... SP apunta darrera ocupada
   1. Push: Pugem SP... guardem dada SP = SP – 1 ... ADREÇA 🡪 [SP]
   2. Pull: Recuperem dada... baixem SP [SP] 🡪 ADREÇA (SP) ... SP = SP + 1
3. Apilem cap a baix... SP apunta primera lliure
   1. Push: Guardem dada... baixem SP ADREÇA 🡪 [SP] ... SP = SP + 1
   2. Pull: Pugem SP... recuperem dada SP = SP – 1 ... [SP] 🡪 ADREÇA (PC)
4. Apilem cap a baix... SP apunta darrera ocupada
   1. Push: Baixem SP... guardem dada SP = SP + 1 ... ADREÇA 🡪 [SP]
   2. Pull: Recuperem dada... pugem SP [SP] 🡪 ADREÇA (PC) ... SP = SP – 1

Imagen que contiene Tabla

Descripción generada automáticamente

**Gestió PC / Crides a funció: Stack Pointer (SP).  
Registre 16 bits (SRAM Memòria 1Byte/Posició)**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Stack a SRAM... Memòria 1 byte ample...

Dades... Operands... 1 byte... 1 posició...

Adreces... de memòria de codi... 2 bytes... 2 posicions...

**Gestió PC / Crides a funció: Stack**

Per tant, a Stack guardem les adreces de retorn a les crides a funció **(CALL/RET)**

**De fet podem guardar qualsevol dada que necessitem poder recuperar més tard**

Les **variables locals** de les funcions també les posarem a la pila (les declarem a la pila)

Imagen que contiene Gráfico

Descripción generada automáticamente

A mida que vaig creant variables, les vaig apilant...

Cada variable amb la mida que li correspongui

Les **variables locals** de les funcions també les posarem a la pila

Imagen que contiene Gráfico en cascada

Descripción generada automáticamente

A mida que vaig creant variables, les vaig apilant...

Per tenir la referència de on comencen les variables locals farem servir un punter **(Frame Pointer)**

Imagen que contiene Gráfico

Descripción generada automáticamente

La posició de memòria on està la primera variable estarà apuntada per **FP** que no es mou al llarg de la funció: abans de crear variables **FP = SP.**

Com podem anidar crides, també haurem de guardar FP a Stack abans de fer una crida a una funció.

Diagrama

Descripción generada automáticamente con confianza media

Per tant, a Stack guardem:

1. FP abans de fer crida a funció
2. Adreça de retorn de la crida a la funció (copiem SP 🡪 FP)
3. Variables locals

A mida que guardem informació per recuperar més tard, SP va decrementant el seu valor.

Si tenim moltes crides a funció anidades... si tenim moltes variables locals...

La **pila quedarà desbordada**... (quan passarà això?) 🡪 **STACK OVERFLOW!**

**Blocs de memòria a SRAM**

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

**Apartat Teoria del e-Campus**

**1. Introducció**

**A. Característiques Microprocessador – Microcontrol·lador**

**Definicions**

**Components**: Elements addicionals integrats en MC, elements addicionals externs en MP

**Característiques de memòries**: Volàtil, no volàtil, RAM, ROM, PROM, EPROM, EEPROM, NVRAM

**Potència** (Nuclis, operacions...): Més en MP que en MC

**Velocitat** (Cicles): Molt més en MP que en MC

**Consum** (Gestió de bateria): En MC és un element clau, no sempre hi ha facilitat de càrrega. Mode Standby.

**Temperatura**: Els MC tnen un rang de temperatura (marge dinàmic) molt més ample que MP. Aguanten temperatura més baixa i més alta

**Memòria**: Els MP tnen molta més memòria amb la que poden treballar. El MC en té molta menys.

**Número d’instruccions**: Més en MP (MPU) que en MC (MCU)

MPU 🡪 Orientació global, MCU 🡪 Orientació específica

**Tipus d’instruccions**: MCU uniformes, MPU simples/complexes, grans/petites, ràpides/lentes

**Tipus d’aplicacions**: MPU alta demanda de capacitat de procés, MCU tasques molt específiques/concretes

MPU 🡪 Entorn multitasca, MCU 🡪 Entorn mono tasca

**Mida**: MPU gran, no és fonamental, limitació per altres factors, MCU petita, és un requeriment que sigui el més petit possible

**Capacitat I/O**: MPU no integrades, MCU analògiques i digitals integrades

**Emissions electromagnètiques**: més protegit i menys emissions en MCU (entorns industrials)

**Rang de temperatura**: Més ampli en MCU

**Velocitat engegada**: Més ràpid en MCU

**B. Solucions de connectivitat**

**1- Conceptes bàsics**

**Comunicacions paral·lel**: Poca distància, ràpides.

**Comunicacions sèrie**: Molta distància, més lent perquè seqüencialitza la informació.

* **Comunicacions sèrie síncrona**: Enviament de blocs, trames, paquets, frames... Més ràpida, més senzilla. Interval de transmissió constant, requereix rellotge precís.
* **Comunicacions sèrie asíncrona**: Enviament de caràcters / bytes... Més lenta, menys senzilla. Interval de transmissió no constant, menys eficiència de canal.

**Serial·lització**: Comunicació paral·lel-sèrie-paral·lel

**MSB**: Most Significant Byte --- **LSB**: Less Significant Byte

**2- UART**

També existeix una solució USART: amb possibilitat de comunicació síncrona.

UART 🡪 Universal Asynchronous Reciever / Transmitter

* Serialitza dades
* Comença per LSB
* Bit de inici/parada i paritat
* Comunicació asíncrona
* Comunicació 1:1 (entre dos)
* Comunicació halfduplex
* Distàncies curtes
* Comunicació amb dos línies
  + MOSI (Master Output Slave Input) com a transmitter
  + MISO (Master Input Slave Output) com a receptor
* Comunicació TX a RX (MOSI-MISO)
* Connexió GND entre dispositius
* Mida de trama màxim 9 bits

**3- SPI**

SPI 🡪 Serial Peripheral Interface

* Comunicació asíncrona
* Intercanvia packets
* Comunicació Master-Slave (transmissor-receptor, server-client...)
* Master acostuma a ser MCU, Slave acostuma ser memòria, sensor, display
* Relació 1:N
* 4 línies
  + MOSI: Enviament de dades de Master a Slave: MOSI (Master) a MOSI (Slave)
  + MISO: Enviament de dades de Slave a Master: MISO (Slave) a MISO (Master)
  + SCLK: Serial Clock, clock enviat de master a slave
  + SS/CS: Slave Select o Chip Select, per la selecció de Slave amb qui es vol parlar: senyal passa de High a Low i novament a High per activar.
* La selecció de Slave pot fer-se amb una línia SS/CS per Slave, o amb una línia compartir de Daisychain
* Més velocitat que I2C
* Comunicació fullduplex
* Només admet 1 Master

**4- I2C**

I2C 🡪 Inter-Integrated Circuit

* 2 línies
  + SDA: Serial Data per intercanvi d’informació (dades)
  + SCL: Serial Clock per enviament de clock de Master a Slaves
* Cada Slave té una adreça de 7 bits (Identificació) + 1 bit (R/W)
* Permet múltiples Masters/Slaves
* ACK/NACK de cada trama
* Hardware més senzill que UART, més complicat que SPI
* Comunicació halfduplex
* Block de inici/final de comunicació entre Master Slave
* Taxa menor que SPI

**2. Execució d’instruccions**

**A- Informació prèvia**

**Màquina d’estats:**

* SETUP: Per configuració I/O
* LOOP: Pel cicle d’execució del programa (recurrent i únic)

**Edianness:**

* Ordre en el que s’escriuen els número de més d’un byte, depèn del micro, del SO...
  + **Big Endian**: d’esquerra a dreta
  + **Little Endian**: de dreta a esquerra
* **Mètriques**:
  + **MIPS**: Milions d’instruccions per segon
  + **MFLOPS**: Milions d’instruccions en coma flotant per segon

**Tipus processadors**

**1- Classificació 1: En base a conjunt instruccions**

**CISC (Complex Instruction Set Computer)**

* Conjunt gran instruccions
* Instruccions complexes / simples
* Instruccions llargues / curtes
* Instruccions ràpides / lentes

**RISC (Reduced Instruction Set Computer)**

* Conjunt reduït d’instruccions
* “Totes” les instruccions uniformes: simples, curtes, ràpides (1 cicle de rellotge)

**2- Classificació 2: En base a número de bus**

**Arquitectura de Von Neumann / Princeton**:Un bus, memòries iguals

**Arquitectura Harvard**: Dos busos, més ràpida

**B- Cicle d’instruccions**

**Procés d’execució de programes**

1. **Cicle de FETCH**: Per llegir la instrucció de memòria.

2. **Cicle de DECODE**: Per interpretar (descodificar) la instrucció.

3. **Cicle d’EXECUCIÓ**: Per la execució de la instrucció a la ALU.

4. **Cicle de STORE (WRITE BACK)**: Per la escriptura de resultats en memòria (dades).

Per tal de saber quina és la següent instrucció a agafar tinc **PC = Program Counter = Comptador de Programa**.

Indica la posició de memòria d’on la unitat de FETCH ha d’agafar la següent instrucció.

Aquest comptador s’ha d’anar modificant després de cada operació de FETCH, per defecte es mou a la següent instrucció: execució seqüencial de programa.

**Execució seqüencial**

La seqüència d’execució és F1-D1-X1-S1-F2-D2-X2-S2

**Execució pipelining**

Si totes les instruccions són igual de llargues (trigo el mateix de fer fetch, decode, execute i Store) puc paral·lelitzar.

Tinc 4 unitats de procés:

1. Unitat de FETCH que farà FETCH

2. Unitat de DECODE que farà DECODE

3. Unitat de EXECUTE que farà EXECUTE

4. Unitat de STORE que farà WRITE BACK

El micro pot llavors acabar fent (en el límit) n instruccions en n cicles d’instrucció/rellotge.

Per què el pipelining sigui eficient / fàcil:

* Necessito tenir dos busos: un de codi per poder fer FETCH al mateix temps que en el bus de dades faig la operació de STORE
* Necessito que el micro sigui RISC i totes les instruccions siguin uniformes

**Problemes en micros CISC**

* Exemple amb operacions de FETCH de duració variable: 1 cicle, 2 cicles...
* Exemple amb operacions de DECODE de duració variable: 1 cicle, 2 cicles...
* Exemple amb operacions de FETCH-DECODE de duració variable: 1 cicle, 2 cicles...
* Exemple amb operacions de FETCH-EXECUTE de duració variable: 1 cicle, 2 cicles...

**Superescalabilitat**

Capacitat de diposar de més d’una unitat de FETCH i/o DECODE i/o EXECUTE i/o STORE

Molt més fàcil poder fer pipelining llavors en micros CISC

**Dependència**

No poder continuar amb el cicle d’execució per manca d’alguna dada

1. Dependència estructural: manca unitat estructural per fer l’acció.

2. Dependència de dada: la dada no està encara disponible: encara no ha acabat la operació anterior.

3. Dependència d’escriptura: es vol escriure i llegir al mateix temps una dada.

4. Dependència condicional: el programa té una condició que fa que el cicle d’instrucció deixi de ser seqüencial, i per tant s’ha de reinicialitzar el procés de pipelining.

**C. Equació de rendiment**

Mesura d’eficiència de micro

R = Temps / Programa = Temps / Cicle \* Cicles / Instruccions \* Instruccions / Programa

Element 1: Temps / Cicle: més eficient (optimitzat) en CISC per freqüència rellotge més ràpida

Element 2: Cicles / instruccions: més eficient (optimitzat) en RISC on val 1. En el cas sempre és més gran que 1

Element 3: Instruccions / programa: més eficient (optimitzat) en CISC que té instruccions més complexes i per tant el programa en pot tenir menys

**3. ATMega328P**

* Anàlisi datasheet
* MCU de 8 bits, high performance, low power, risc avançat
* 131 instruccions (1 cicle per instrucció la majoria)
* 32 x 8 registres de propòsit general
* Fins 20MIPS a 20MHz
* Multiplicador (2 cicles)
* Memòries:
  + FLASH: 32K, no volàtil, codi
  + EEPROM: 1K, dades que volem garantir que són persistents
  + SRAM 2K: variables, informació temporal
* Comptadors/temporitzadors: 2 de 8 bits, 1 de 16 bits, 1 extern
  + Entendre diferències-similituds comptadors/temporitzadors
* 6 canals PWM (entendre concepte duty-cycle)
* 6/8 canals ADC 10 bits (entendre concepte 10 bits)
* Comunicacions sèrie (USART, SPI, I2C)
* Alimentació
  + Entendre concepte Brown-out detection
  + Entendre concepte Black-out
  + Entendre concepte modes de baix consum (sleep modes) (en té 6)

**4. Funcionament intern**

**A- Registres**

32 registres de propòsit general de 8 bits /R0-R31) (Posicions memòria 0x0000-0x001F)

3 registres de propòsit general de 16 bits (RX, RY, RZ)

* RX = R26 - R27
* RY = R28 - R29
* RZ = R30 - R31

**B- Memòria FLASH: Instruccions**

16Kx2

Adreçament amb 14 bits

PC (Registre de 16 bits dels quals els dos primers sempre seran 0) en necessita només 14

Primera posició de memòria 0x0000

Primera posició de memòria 0x3FFF

**C- Memòria SRAM: Dades (variables)**

2Kx1

Adreçament 11 bits

Primera posició de memòria 0x0000

Darrera posició de memòria 0x07FF = SRAMEND

**D- Registres (per sobre de la SRAM)**

* 32 registres de propòsit general 0x0000-0x001F
* 64 registres de I/O
  + Adreçament relatiu 0x0000-0x003F
  + Adreçament absolut 0x0020-0x005F
* 160 registres de I/O extesos
  + Adreçament absolut 0x0060-0x00FF
* SRAM 0x0100-0x08FF
  + 0x08FF = RAMEND

**E- EEPROM**

1Kx1

Primera posició: 0x0000

Darrera posició: 0x03FF

**F- Program Counter**

Per gestionar la propera posició de memòria de la que s’ha de fer FETCH.

1. Com gestionem les crides a funcions

La posició de retorn la guardarem a **stack** on les anirem apilant / desapilant

**G- Stack Pointer**

Registre per tenir la posició en la que estem treballant de la pila (stack)

Cada vegada que fem una crida a una funció guardarem (apilarem) l’adreça de retorn a la pila

Cada vegada que retornem d’una crida a una funció (ret) recuperarem (desapilarem) l’adreça de retorn de la pila

* Les operacions d’apilar / desapilar es poden fer en ordre ascendent (primer posicions de memòria més baixes i després més altes) o descendent (al revés)
* Les operacions es poden fer apuntant a la primera posició lliure o a la darrera posició ocupada

Dues operacions bàsiques:

* Apilar: **PUSH**
* Desapilar: **POP**

Opcions:

* SP apunta a primera posició lliure i apila de posicions altes a baixes

Diagrama

Descripción generada automáticamente con confianza baja

* SP apunta a darrera posició ocupada i apila de posicions altes a baixes

Imagen que contiene Gráfico de cajas y bigotes

Descripción generada automáticamente

* SP apunta a primera posició lliure i apila de posiciones baixes a altes

Imagen que contiene Diagrama

Descripción generada automáticamente

* SP apunta a darrera posició ocupada i apila de posicions baixes a altes

Imagen que contiene Diagrama

Descripción generada automáticamente

En el cas ATMega328P: Funcionament SP apunta a primera posició lliure i apila de posicions altes a posicions baixe sen arrencar programa SP = 0x8FF

En cas ATMega328P si guardem adreces de memòria increments / decrements SP de 2 en 2 (adreça és de 2 bytes).

Sempre que vulguem guardar una dada que volem poder recuperar més tard fem push per guardar i pop per recuperar.

**H- Frame Pointer**

Registre per tenir la posició en al que comencen les variables locals a una funció (local) a la pila (stack)

Quan entrem en una funció: primera funció FP=SP

A partir d’aquí es desclaren les variables locals (per les que es guarda espai a l’stack).

Per accedir a les variables locals, el punter de referència és frame pòinter.

Com només tenim un registre FP, haurem de guardar el seu valor a la pila quan tinguem les funcions anidades (call anitat).

Per tant, abans de fer la crida a la funció guardem el valor de SP, després adreça de retorn, després clavem FP,...

**I- Paràmetres d’entrada/sortida**

Podem passar paràmetres per registre o per stack.

En el cas de MCU, ho farem per registres si el número de paràmetres és menor al màxim permès. En el cas de MPU ho farem per stack.

**J- Paràmetres d’entrada per stack**

La funció que fa la crida guardarà els valors dels paràmetres d’entrada a la pila després de guardar FP, i adreça de retorn abans de clavar FP.

Els paràmetres d’entrada es guarden de dreta a esquerra: primer PAR3, després PAR2 i finalment PAR1 si tenim tres paràmetres.

Per tant a la pila tindrem:

FP1 🡪 RET2 🡪 PAR3 -- PAR2 -- PAR1 -- FP = SP 🡪 VAR1 -- VAR2 -- VAR3

De les opciones que tenim:

* Opció 1: PAR-FP-RET-VAR
* Opció 2: FP-PAR-RET-VAR
* Opció 3: FP-RET-PAR-VAR
* Opció 4: FP-RET-VAR-PAR

La opció 4 no és viable.

De les altres 3, la utilitzada és la opció 3 (per rendiment).

**K- Paràmetres de sortida per stack**

La funció que fa la crida guardarà espai pel paràmetre de retorn (com a màxim n’hi haurà un) (si n’hi ha guarda sinó no).

On? Abans de fer res:

Espai retorn – FP – RET – PAR2 – PAR1 – VAR1 – VAR2

**L- Paràmetres d’entrada per registres cas ATMega328P**

Màxim 9 paràmetres d’entrada

Màxim registres R8-R25

Si son de 1 Byte: R24, R22...

Si són de 2 Bytes: R24-R25, R22-R23

Si són de 4 bytes: R22-R23-R24-R25

...

**M- Paràmetres de sortida per registres cas ATMega328P**

Màxim 1 paràmetre de sortida

Si 1 Byte: R24, R25 = 0

Si 2 Bytes: R24, R25

...