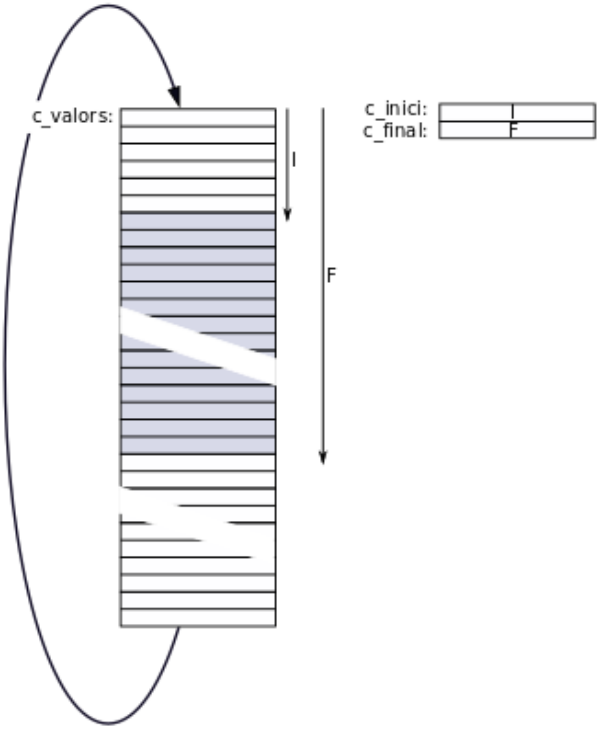
Dispositius Programables

Durada 2:30h

Final - Gener 2021

1. L'objectiu d'aquest exercici és implementar una cua en assemblador. Per aconseguir-ho definirem una estructura de dades aprofitant la memòria de dades i unes operacions sobre aquesta estructura de dades. Es defineix la constant **C_MIDA** com la mida que tindrà aquesta cua. Aquest valor no podrà ser superior a 255. Es defineix un espai de **C_MIDA** bytes a memòria de dades identificat a partir de l'adreça (etiqueta) **c_valors**. També es defineixen les variables de 1 byte **c_inici** i **c_final**. La variable **c_final** és l'índex que apunta a la primera posició disponible dins l'espai de memòria reservada respecte a l'adreça **c_valors**. Es considera que les anteriors estan ocupades. La variable **c_inici** és l'índex que apunta a la primera posició ocupada dins l'espai de memòria reservada respecte a l'adreça **c_valors**. Es considera que les anteriors estan lliures. Per entendre això, la idea clau a l'hora d'implementar aquesta cua és que es fa circular. Això vol dir que quan s'arriba a la última posició de memòria de l'espai lineal definit per **c_valors**, la següent posició torna a ser la primera. Amb aquestes condicions, quan no hi ha cap element a la cua, **c_inici** no pot apuntar enlloc, però es pacta que sigui igual a **c_final**.



   a) Reserva a memòria de dades els bytes corresponents a les variables **c_inici**, **c_final** i **c_valors**. Especifica a quina secció queden definides. Aquestes variables quedaran definides per les etiquetes del seu mateix nom, indicant l'adreça on es troben.

   b) Una operació auxiliar necessària es la macro **inc_mod**. Aquesta macro incrementa en 1 unitat el registre r16 però en aritmètica mòdul **C_MIDA**. Això vol dir que si r16=(**C_MIDA**-1) i cridem a la macro **inc_mod** el resultat serà r16=0. En qualsevol altre cas el resultat serà r16+1. Implementa aquesta macro utilitzant el mínim nombre d'instruccions possible.

   c) En quan a les operacions definides sobre aquesta estructura de dades, en primer lloc tenim **c_encua**. Aquesta subrutina realitza l'operació d'encuar a la cua i s'implementa escrivint el

valor que se li passa coma a paràmetre pel registre r16 a la posició de memòria de dades indicada per l'adreça de **c_valors** + l'índex **c_final**. Un cop fet això, s'incrementa el valor de l'índex **c_final** en 1 unitat tenint en compte l'aritmètica modular de base **C_MIDA**. Teniu disponible la macro **inc_mod** per si us interessa. Implementeu aquesta subrutina.

d) La següent operació disponible serà **c_desencua**. Aquesta subrutina realitza la funció de desencuar de la cua. S'implementa llegint el valor de memòria de dades definit per l'adreça de **c_valors** + l'índex **c_inici** i retornant aquest valor pel registre r16. Al igual que en el cas anterior, s'incrementa el valor de l'índex **c_inici** en 1 unitat en aritmètica modular de base **C_MIDA**. Justifica si la subrutina **c_desencua** ha d'esborrar el valor contingut a la posició de memòria **c_valors** + **c_inici** un cop s'ha llegit i implementeu la subrutina.

e) Es vol disposar també de la subrutina **c_init**. Aquesta subrutina inicialitza la cua deixant una cua totalment buida. Implementa aquesta subrutina.

f) La següent operació és de consulta. S'ha de definir la subrutina **c_esbuida** que ens retorna en el flag Z si la cua està buida. Z serà 1 si està buida i Z serà 0 si no està buida.

g) També es vol disposar de la subrutina **c_esplena**. Retornarà en el flag Z un 1 si està plena i un 0 si no està plena. La implementació que us demana el vostre superior a la feina consisteix en comprovar si l'increment modular de 1 unitat de **c_final** coincideix amb **c_inici**. En cas afirmatiu és que està plena. Al ser una ordre del vostre superior implementeu aquesta subrutina d'aquesta manera. Però també justifiqueu si està cometent algun error a la seva implementació o no.

2. A partir de la cua implementada de l'exercici anterior, es vol millorar el comportament de la pràctica 9 (Generador Morse) utilitzant aquesta cua. En termes generals, l'estructura del codi que heu implementat a la pràctica 9, sense totes les macros/subrutines i directives auxiliars necessàries, és la següent:

```
__vector_18:
        PUSH r16
        LDS r16, UDR0
        CALL busca_codi
        CALL genera_morse
        POP r16
        RETI

main:
        inicialitzacions
        SEI
bucle:
        RJMP bucle
        RET
```

On **inicialitzacions** és la macro/codi encarregat de configurar el sistema i perifèrics. **busca_codi** és la subrutina que transforma el codi ASCII rebut pel port sèrie en el codi morse. En aquesta subrutina, el registre r16 es fa servir tant per passar el paràmetre en codi ASCII com per retornar el codi morse. **genera_morse** és la subrutina que a partir del paràmetre existent a r16 va generant seqüencialment i progressivament el codi morse corresponent. Aquesta última subrutina tarda molt temps en executarse, ja que segueix la pauta de punts, ratlles i silencis en base a una unitat de temps. Pel fet d'estar dins de la interrupció de RX de port sèrie, provoca que totes les interrupcions no puguin ser ateses durant molt temps. L'objectiu és utilitzar la cua definida a l'exercici 1 per evitar aquest problema.

a) Reescriu l'estructura anterior afegint pseudocodi i aprofitant les operacions sobre la cua definides a l'exercici 1.

b) Transforma el pseudocodi en codi assemblador. Aquest apartat depèn de l'anterior i només es tindrà en compte si l'apartat a) està correctament resolt.

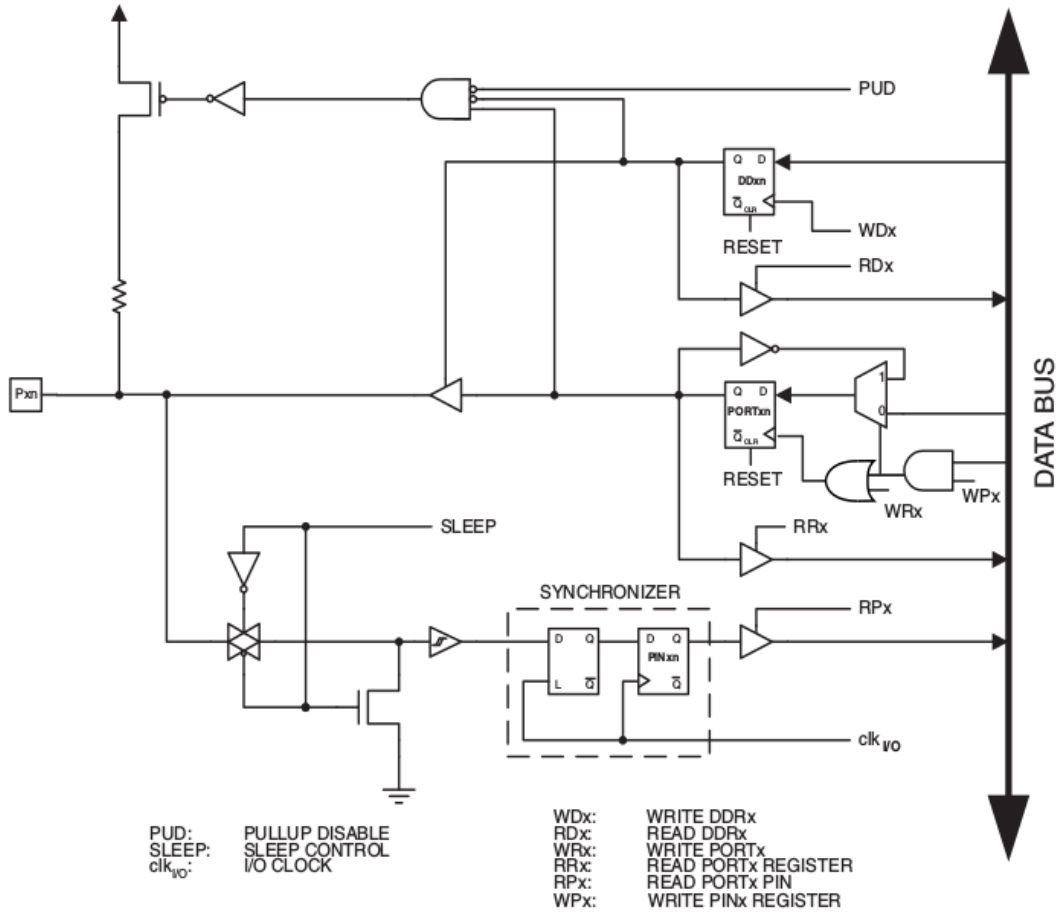
3. Es disposa d'un registre REG de 4 bytes format pels registres r3:r2:r1:r0. Es valorarà la utilització mínima de recursos (registres, instruccions, temps d'execució).

   a) Dissenyeu la macro **sbitr** de manera que desplaça a la dreta 1 bit aquest registre REG. Considereu que el nou bit d'entrada es troba al flag C i que el bit sortint es deixa al mateix flag C.

   b) Considereu que aquest registre REG implementa un valor enter amb precisió de 4 bytes. Es vol implementar l'operació **div2u** i **div2s** que divideix per 2 el valor d'aquest registre. **div2u** considera que REG conté un valor sense signe i **div2s** considera que REG conté un valor amb signe. Implementeu aquestes dues subrutines. Disposeu de **sbitr** si us interessa.

   c) Dissenyeu la subrutina **sumar16** que sumi el valor de r16 al registre REG i deixi el resultat al mateix registre REG. Considereu que r16 codifica un valor sense signe. Els flags Z,C,H,N,V i S han de quedar correctament actualitzats. Si cal, redissenyeu **sumar16** canviant el nom a **sumar16s** considerant que r16 codifica un valor amb signe. Justifica la resposta en el cas que no sigui necessari.

4. La següent figura mostra l'esquema de les potes i/o de l'AVR. Es valorarà la utilització mínima de recursos per a les implementacions següents.



   a) Escriu la seqüència d'instruccions per definir el Port D amb els bits [7,6] com sortides amb valor 1, els bits [5,4] com sortides amb valor 0, els bits [3,2] com entrades sense pull-up i els bits [1,0] com entrades amb pull-up.

   b) Dissenya una subrutina **lec7531or** que deixa al r16 un valor diferent de 0 si el valor d'alguna de les potes del Port D [7,5,3,1] és 1, en cas contrari deixa el valor de 0.

   c) Dissenya una subrutina **lec6420and** que deixa al r16 un valor de 0 si el valor de totes les potes del Port D [6,4,2,0] són 1, en cas contrari deixa un valor diferent de 0.