**CPS:** cyber.physical system, integra la computazione con i processi fisici (l’uso ottimizzato dell’energia).

I processi fisici eseguiti tutti in una volta, processi software invece eseguiti in modo procedurale (sequential steps)

Misurare e controllare i processi fisici è l’obiettivo principale dei sistemi embedded

I processori embedded hanno una precisa funzione dedica che gli permette di utilizzare meno energia ed avere costi inferiori.

**ISA: instruction set architecture**

Le istruzioni possono essere eseguite rispettando delle regole strutturali, ad esempio la grandezza della word.

intel x86 è un ISA

**Realizzazione di un processore o di un chip:**  Pezzo di silicio

ISA garantisce la portabilità, si può avere la stessa isa su processori differenti.

L’esecuzione logica, grazie alle ISA uguali, è la stessa, mentre il comportamento del sistema può variare a seconda del CPS.

**Microcontrollore**

Circuito integrato che possiede CPU , memoria, device di I/O e timers.

I microcontrollori implementano diverse versioni di arm instruction set.

**DSP: digital signal processor**

Tipicamente sono macchine CISC e includono istruzioni specifiche per supportare filtri FIR.

I processori DSP posso campionare, a seconda del fenomeno fisico (audio, video.. etc), usando diverse frequenze. Si usano i DSP a virgola fissa con una precisione che va dai 16 ai 24 bits.

DSP inizialmente erano programmati in assembly, adesso in C o in altri linguaggi di alto livello (si differenzia per questo dalle architetture RISC).

**FIR**

Un’applicazione dei processori DSP è il filtro FIR. è semplicementi l’implementazione di una sommatoria ( incollare ). Consiste nel moltiplicare ai (coefficiente del filtro) con i campioni x(n-i).

L’idea è quella di calcolare y(n), cioè l’output del sistema al tempo n. Abbiamo bisogno di sapere tutti valori di input. Dobbiamo sapere non solo il campione attuale ma anche quelli precedenti. Un tap è il campione subito precedente a quello all’attuale, il tap comprende anche la ai.

*Se N è dispari e i coefficienti sono simmetrici la formula cambia e si il numero delle moltiplicazioni si riduce.*

La nostra finestra del passato è limitata, in questo caso in 4 campioni. Nel nostro esempio viene fatta la media aritmetica dei valori perché ai = 1/4 ma solitamente si applica la sommatoria sopra perché gli ai cambiano.

**Linguaggio Imperativo**

C è un linguaggio imperativo, quando si usa C, per implementare degli step bisogna utilizzare delle librerie di thread adatte.

Java è un linguaggio ancora più imperativo di C esteso con dei costrutti che supportano direttamente i thread.

**VLIW: very large instruction word & Multi-issue**

Molti mini processori oggi supportano esecuzioni in parallelo, usando istruzioni streams multi-issue (altro tipo di architettura rispetto a VLIW) o architettura VLIW.

I processori con istruzioni streams multi-issue posso eseguire istruzioni indipendenti in modo simultaneo.

L’obiettivo principale di queste architetture è quello di finire il compito nel minor tempo possibile (migliorando le performance).

**Concorrenza**

Le esecuzioni concorrenti hanno un’altra scuola di pensiero, preferiscono infatti eseguire il compito in un preciso istante (es candela per l’accensione del motore a diesel).

Il programma concorrente può essere convertito in uno stream sequenziale da un sistema operativo e tornare al programma concorrente tramite hardware, quest’ultima traduzione migliora le performance.

**Pipelining**

è un macchina semplice **RISC.**

La maggior parte di architetture prima dell’avvento delle **RISC** consentivano di piazzare dati a qualsiasi indirizzo. LE **CISC** hanno una logica più intricata

Questo è un modello di cache, prende dati dalla RAM e li elabora qui dentro.

Nei sistemi embedded si hanno due memorie separate, una per i dati e una per le istruzioni, senza memorie cache. Solitamente la memoria delle istruzioni non ha necessità di essere scritta più volte. Le barre grigie sono dei registri, alcuni dei quali potrebbero essere rimossi ma comunque la struttura non cambia e i dati devono fluire sempre da sinistra verso destra.

Si effettuano le seguenti operazioni in seguenza:

**FETCH**

1. Carico le istruzioni dalla locazione di memoria dell’instruction memory indacata dal PC
   1. C’è anche la logica che gestisce incremento del PC
2. Il 4 è presente perché questa architettura accetta word di 4 byte e quindi gli indirizzi di memoria saranno tutti multipli di 4.
3. Il PC attuale va direttamente nell’ instruction memory (istruzione puntata al PC)
4. Dalle instruction memory escono sempre (in questo caso) 32 bit, e saranno poi decodificati nella fase di decode.

**DECODE**

1. dei 32 bit due operandi vanno a finire nel register bank e i restanti nel decode.
2. Dal register bank possono uscire due valori che vanno nello stato di execute

**EXECUTE**

1. Troviamo ALU che riceve in input i valori del register bank e fa i conti.
2. Dopo che l’ALU elabora i dati se stiamo usando una macchina CISC (ha 2 registri sorgente di cui uno è quello di destinazione ) il registro di destinazione coincide con quello sorgente e il dato dopo essere stato elaborato viene riscritto nel register bank. Nelle macchine RISC invece ci sono 3 registri (2 solo per sorgente e 1 solo destinazione), uno dei quali è usato per salvare il dato elaborato (quello di destinazione).
   1. NB le macchine RISC non presentano data memory (**caratteristica principale**)

**MEMORY**



In questo tipo di pipeline riscontriamo il problema della sovrapposizione delle istruzioni (hazard) che viene risolto ritardano l’istruzione successiva fino a quando l’istruzione antecedente non ha concluso il writeback.

**CISC: complex instruction set computer**

Un processore con un istruzioni complesse

**RISC: reduced instruction set computers**

**INNER LOOP DI UN FIR:**

RPT numberOfTaps - 1

MAC \*AR2+, \*AR3+, A

I registri AR2 e AR3 possono essere usati per implementare un buffer circolare.

**MMX**

Sono tecnologie integrate su l'Intel Pentium che introduce il parallelismo tra subword (particolare forma di vector processing, cioè set di istruzioni che includono operazioni su più elementi di dati contemporaneamente).

……

I processori superscalar hanno lo svantaggio di non poter gestire interrupt e threads e hanno difficoltà nel predire il tempo di esecuzione.

**FPGA: field programmable gate array**

Sono chip che contengono hardware programmabile tramite strumenti di design.

Ad esempio i soft core sono processori implementati negli FPGA.

…………………………………………………………………………………………………………..

**ARM**

I registri ad uso generale sono 13 (Da R0 a R12)

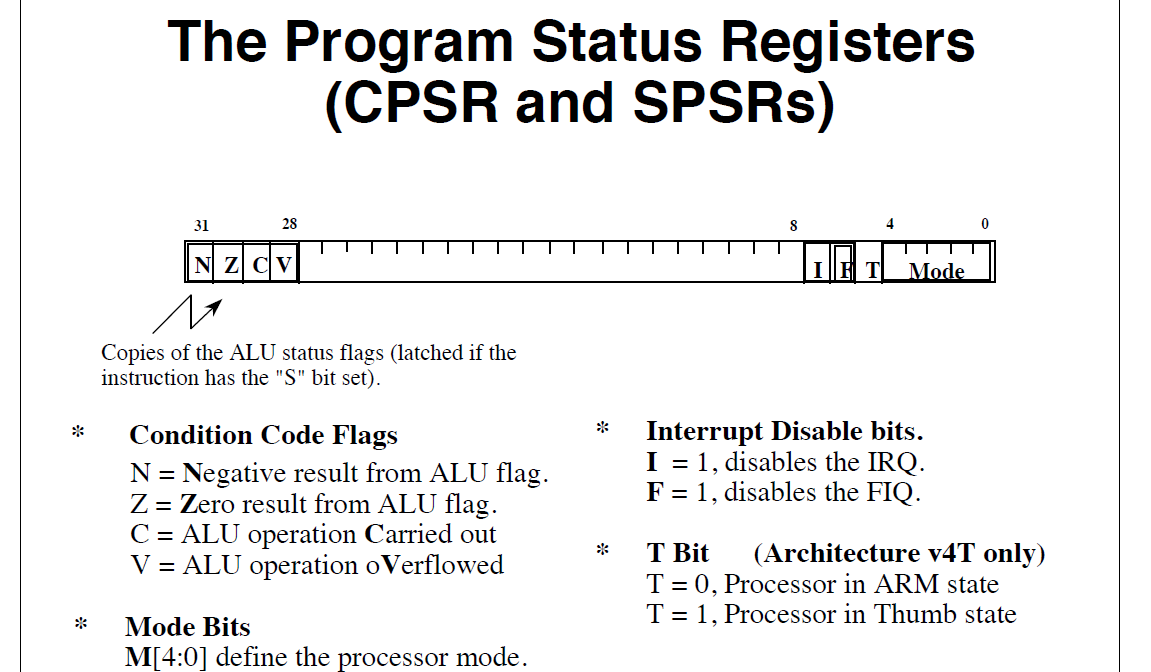
R13 viene usato come stack pointer

R14 link register, all’interno di questo registro viene salvato ( dopo l’esecuzione di BL) il contenuto di R15

R15 all’interno vi è il program counter

**CPSR : Current Program Status Register**

* 4 bit esprimono altrettante condizioni ( Negative, Carry, Zero e Overflow);
* Il bit T distingue tra due (super) modalità di funzionamento :
  + Quella usuale (ARM)
  + E la modalità Thumb
* I bit I ed F Abilitano le interruzioni normali (I) e veloci (F)
* I bit M4-M0 identificano il modo di funzionamento



**The program counter (R15)**

Tutte le istruzioni sono lunghe 32 bits

Tutte le istruziono devo avere la stessa lunghezza

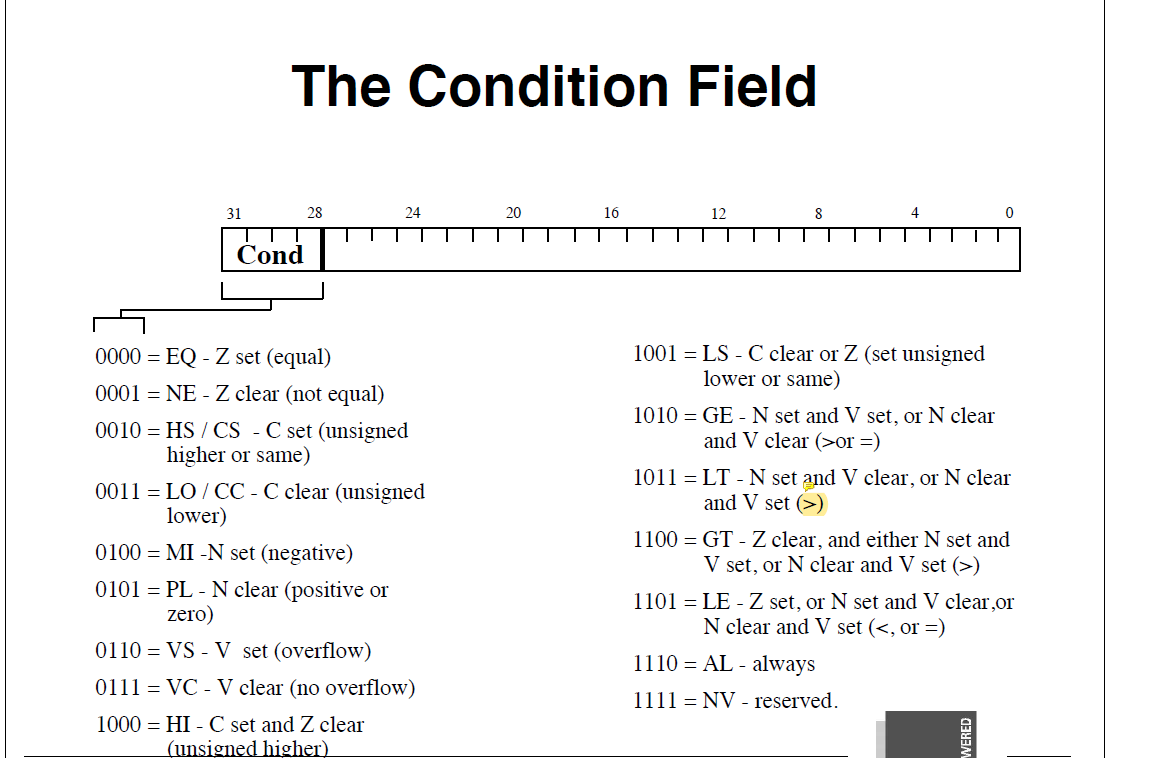
Il valore del program counter è memorizzato nei bit [31:2] e i bit 1 e 0 sono settati a zero.

**Which mode, or modes has the fewest available number of registers**

**available? How many and why?**

-----

Le istruzioni condizionali, anche se richiedono una penalità di tempo, sono preferibili perché in ogni caso vengono eseguite in un tempo minore rispetto ai branches o subroutine.



**Branch instructions**

Il branch cambia il contenuto del program counter e determina quale istruzione sarà eseguita. L’istruzione a cui saltare viene determinata sommando un valore costante (con segno) al PC, quest’ultimo punta all’istruzione che si trova due word dopo.

L’offset è contato in word e quindi l’offset prima di essere sommato al PC viene moltiplicato per 4, ottenendo un offset di 26 bit. Abbiamo un range di salto di 32 MB in avanti e 32MB indietro.

Branch non tiene memoria e branch with link si.

**Data processing Instructions**

ARM è un’architettura load /store, istruzioni che lavorano con i registri e non con la memoria.

**Operando 2 del barrel shifter**

L’operando 2 è grande 12 bit e tra i 12 bit ci sono le informazioni inerenti agli shift e alle rotazioni. Lo shift può essere di 5 bit se abbiamo numeri unsigned.

Con un registro possiamo decidere di quanto shiftare un altro registro.

**Per il valore immediato** il secondo operando è grande 8 bit ( posso ruotarlo a destra di quanto voglio basta che sia pari).

LDR (Load register) è il modo raccomando per caricare i registri in memoria

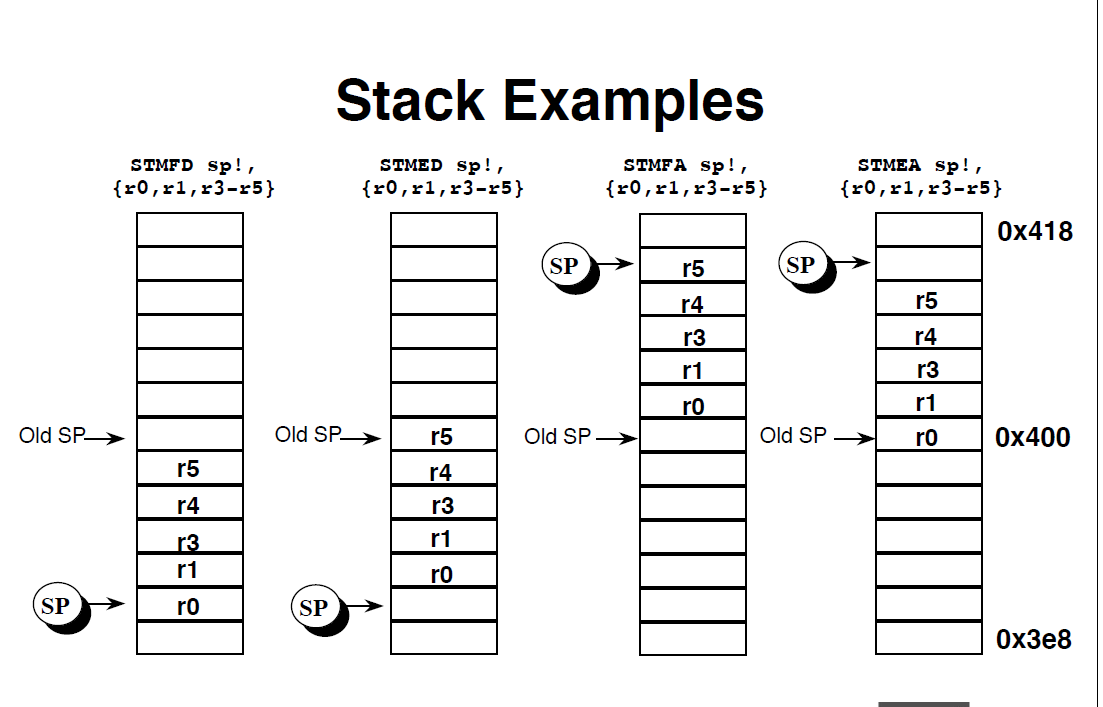
Se si fa una moltiplicazione, **RD = Rm\* Rs**, dove Rs deve essere quel numero che contiene più 0 nei MSB, così che si possa usufruire della terminazione anticipata.

**Pre (**prima che il trasferimento venga effettuato) **e Post (**Dopo l’avvenuto trasferimento) **addressing**

Post index addressing causa l’auto incremento del registro base.

**Operazione di PUSH**

**STMFD sp !**

****

**OPERAZIONE POP**

**LDMFD SP !, {r0-r12, pc }**

**-----------------------------------**

**Interrupt**

è un istruzione definita per l’utente e si avvia quando occorre un determinato evento.

Durante l’occorrenza dell’eccezione il sistema passa alla modalità supervisor mode, inoltre lo scatenarsi dell’eccezione fa chiamare anche il GESTORE DELLE ECCEZIONI.

**USARE MRS AND MSR**

Servono a trasferire dati da un registro specifico a quello general purpose.

Per trasferire il contenuto del PSR (Program status register, cioè un registro specifico) in un registro generico usiamo MRS.  
Per trasferire un valore aggiornato indietro al PSR si usa MSR

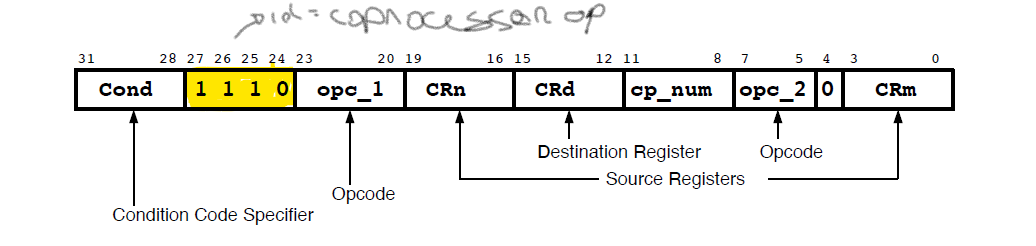
**Arm supporto 16 coprocessori, divisi in**

**3 tipi di istruzioni di coprocessori**

1. Coprocessori per elaborati i dati
2. Coprocessori (to/from ARM) register transfers
3. Coprocessori memory transfers (load and store to/from memory )

Come implementare un coprocessore

* Hardware
* Software
* Entrambi



Nell’arm instruction set le istruzioni vengono estese grazie ai coprocessori.

**GCC (**SOFTWARE TOOLCHAIN**)**

**Diversi modelli di programmazione per programmi embedded:**

* compilation : usa software toolchain per produrre l’eseguibile(es: GCC)
* cross-compilation : usa software toolchain per produrre codice per la macchina target (es: GCC)
* interactive environment : usa un interprete (es: Forth) che può essere implementato per macchine che non hanno un SO (bare-metal)

Calling Convention

* registri da r0 a r3 sono usati per passare valori per argomento nelle subroutine e per ritornare un valore da una funzione
* registri da r4 a r8, r10, r11 sono usati per tenere il valore delle variabili locali delle subroutine
* nel registro r0 ritornato Fondamental Data Type di dimensioni inferiori a 4 byte che sia 0 o con segno esteso
* in r0 sono inoltre ritornati i valori lunghi una word (word size) tipo int, float etc..
* Se invece è di dimensione double word size vengono ritornati in 2 registri (r0 e r1)

Passaggio di parametri

* The base standard provides for passing arguments in core registers (r0-r3) and on the stack. For subroutines that take a small number of parameters, only registers are used, greatly reducing the overhead of a call

**target bootstrap e bootloader**

* Besides internal storage, it may be

more convenient to load startup code

from external storage or even other

machines, for instance those used in

developing software for the target,

through some network connection

* Bootstrapping is most of the time

performed by some bootloader code

installed in the target storage

▸ There exist many types of bootloaders

▸ Often multi-stage bootloaders are used

**Special storage can be reserved to bootloader**

**code by the manufacturer**

▸ SoCs are often provided with a pre-installed

bootloader that eases programming the target

through USB, network, Over-the-Air (OTA)

upgrading

▸ Recovery modes (e.g. Device Firmware

Update or DFU) are based on a boot loader

able to program the target so that it can be

restored to the factory state in case the user

software (or even the OS) gets corrupted

▸ Other debugging and diagnostic modes

can be included in the bootloader

**Sistemi di bootstrap dei target**

JTAG e SWI supportano programmi in bootstrapping e verificano codice in runtime (debugging)

VC (broadcome video-core) carica il codice di bootloader dal file bootcode.bin e lo esegue.

**meccanismo per boot del sistema operativo**

▸ Our target code is obtained through the following steps:

▸ Code is written into a source (.s) file

▸ vi test.s

▸ The assembly file is assembled into an Executable and Linkable Format (ELF) object file (.o)

▸ arm-none-eabi-as test.s -o test.o

▸ The object file is processed by the linker to resolve addresses for a fixed memory model producing a finalized ELF file

▸ arm-none-eabi-ld -T kernel7.ld test.o -o test.elf

▸ The objectdump tool can be used to obtain the finalized ML and assembly code in a listing file (.list)

▸ arm-none-eabi-objdump -D test.o > test.list

▸ The objectcopy tool is used to extract only the ML code from the finalized ELF file in binary format

▸ arm-none-eabi-objcopy test.elf -O binary test.bin

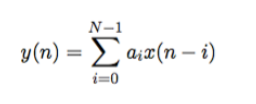
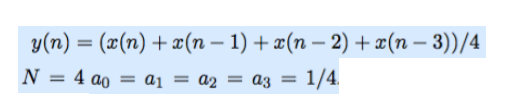
▸ The binary file is renamed to follow the bootstrap conventions

▸ mv test.bin kernel7.img

▸ The image file can then be simply copied onto the microSD to be executed at startup

**RISPOSTE ALLE DOMANDE DI ESAME**

**Cos'è un tap?**

* Un tap è il **campione** subito precedente a quello attuale, il tap comprende anche le ai.
* 
* 
* i TAPS: sono il numero di campioni con cui campiono il filtro ideale, + sono migliore sarà il filtraggio, perchè approssimi meglio il filtro ideale.
* Il TAP consente di estrarre ed elaborare i campioni da una sequenza di campioni

**BUS SPI (serial peripheral interconnect)?**

* Il bus SPI è “**full duplex**” quindi, opzionalmente, i dati possono essere trasmessi e ricevuti allo stesso tempo. Usano MOSI e MISO.
* Può essere single slave o multiple slaves. Quest’ultimo a sua volta si divide in due tipologie: Multiple slave select (ss) e Daisy- chaining.

**Che cos'è la condizione di repeated start I2C?**

* La condizione di **START** si ha quando si vuole iniziare a trasmettere qualcosa e nello specifico quando **SCL** è alto e **SDA** è nel fronte di discesa. Quindi repeated start è una condizione simile a quella di start. Differisce da **START** perché avviena prima di una condizione di **STOP**. è utile quando il master vuole iniziare una nuova comunicazione e non vuole lasciare il bus inattivo.

**La condizione repeater start è in alternativa a cosa?**

* è usata in alternativa al **back-to-back STOP**

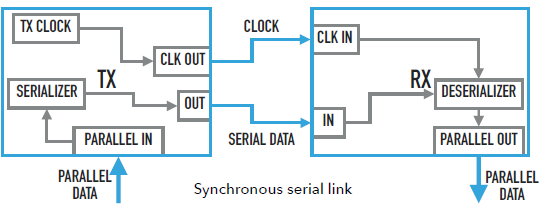
**Perchè ho bisogno di una repeated start?**

* è utile quando il master vuole iniziare una nuova comunicazione e non vuole lasciare il bus inattivo.

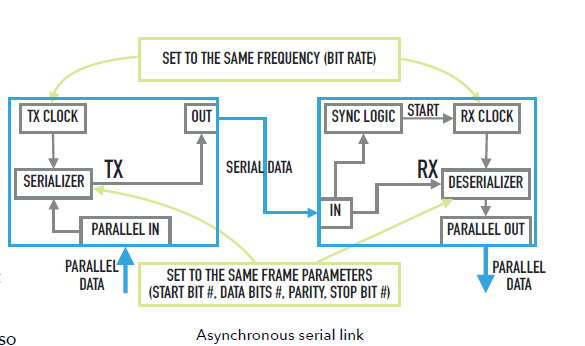
**Come possono essere caratterizzati i bus seriali?**

[connotazioni asincrone, come vengono indirizzati, velocità dei bus, velocità generale i2c SPI]

* I bit serializzati solitamente sono organizzati in una sequenza chiamata **frame**, inoltre possono essere inclusi bit addizionali per la rilevazione di errori o correzioni.
* Il **framing** semplifica:
  + la sincronizzazione tra trasmittente e ricevitore
  + ricostruzione dei dati
  + rilevazione di errori in trasmissione
* Le **connessioni seriali** possono essere:
  + **Sincrone**



* + - tx e rx hanno un clock condiviso
    - Non necessitano di bit di sincronia ma il clock ha bisogno di un’altra linea di trasmissione
    - **Funzionamento** trasmissione:
      * SERIAL DATA e CLOCK sono tenuti alti quando il collegamento è inattiva
      * Quando TX vuole trasmettere un frame, inizializza il clock
      * Ad ogni **fronte di discesa** del CLOCK TX, TX trasmette un bit del frame sulla SERIAL DATA.
      * Ad ogni **fronte di salita** del CLOCK, il deserializzatore in RX campiona un bit dalla SERIAL DATA e lo accoda nel registro PARALLEL OUT
      * SERIAL DATA e CLOCK sono tenuti HIGH fino all’invio del prossimo frame
  + **Asincrone**



* + - La trasmissione dei dati può avvenire in qualsiasi momento
    - Trasmettitore e ricevitore usano ognuno un clock proprio, pre configurati alla stessa frequenza (bit rate).
    - La grandezza del frame deve essere conosciuta sia dal tx che dal rx, in questo modo è richiesta solo la sincronizzazione all’inizio.
    - **Funzionamento** trasmissione:
      * Quando TX vuole trasmettere un frame, invia il bit start (LOW)
      * SYNC LOGIC in RX riconosce la transizione HIGH-LOW(**fronte di discesa**) come condizione di start e avvia il clock
      * In ogni impulso del clock TX, il serializzatore in TX trasmette un bit del frame contenuto nel registro PARALLEL IN attraverso la SERIAL DATA
      * Ad ogni **fronte di salita** del CLOCK RX, il deserializzatore in RX campiona un bit dalla SERIAL DATA e li accoda nel registro di PARALLEL OUT
      * Alla fine del frame, TX invia il bit di STOP (HIGH) e RX riconosce la fine della trasmissione.
      * SERIAL DATA è tenuto alto finchè non è inviato il prossimo frame

**Su i2c che bit rate abbiamo?**

* Il bus i2c è un bus sincrono, multi-master, multi-slave.
* Ogni dispostivo è connesso al bus con l’indirizzamento software è ha un unico indirizzo
* **Non ha un baud rate fisso** richiesto ( NB: il clock è generato dal master)

Il bit rate dipende dalle **modalità**:

* **Standard** modo 100 kbit per secondo (standard mode)
* **Fast modes** (400 kbit/s)
* **High speed mode** (3.4 Mbit/s) con una logica addizionale
* **Fast mode plus** ( 1 Mbit/s) senza logica addizionale
* **Ultra fast mode** UFM (5 Mbit/s)

**Operazione push e pop in arm ?**

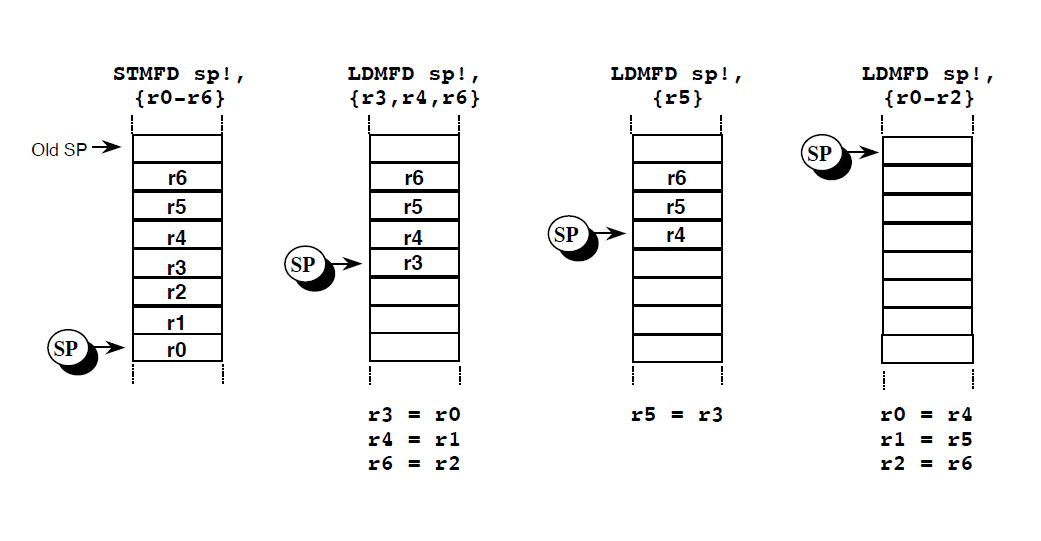
L’ ARM utilizza un FULL DESCENDING STACK, quindi:

L’operazione di **push** viene fatta così:

* **STMFD** sp !, { r0, r1, r3-r5 }
  + Questa operazione quindi inserisce nello stack i registri dall’alto al basso, quindi r5 sarà al top dello stack, lo stack pointer quindi punta ad r0.
  + Se si dovesse fare un’altra operazione di push avremmo sullo stack, dall’alto al basso : r5, r4, r3, r2, r1, r0, r7, r6 ) e l o stack pointer punterà ad r6.

L’operazione di **pop** viene fatta così:

* **LDMFD** sp !, {r0, r1, r3-r5}
  + Questa operazione quindi rimuove i valori dello stack dal basso all’alto. Lo stack pointer quindi punterà al primo elemento da rimuovere.



Alternative

STMFD sp!,{r0-r12, lr} stack all registers and the return address

LDMFD sp!,{r0-r12, pc} load all the registers and return automatically

**Caratteristica processore DSP (filtraggio fir)?**

Tipicamente sono **macchine CISC** e includono istruzioni specifiche per supportare filtri FIR.

I processori DSP posso campionare, a seconda del fenomeno fisico (audio, video.. etc), usando diverse frequenze. Si usano i DSP a virgola fissa con una precisione che va dai 16 ai 24 bits.

**DSP**

inizialmente erano programmati in assembly, adesso in C o in altri linguaggi di alto livello (si differenzia per questo dalle architetture RISC).

**FIR**

Un’applicazione dei processori DSP è il filtro FIR. è semplicementi l’implementazione di una sommatoria ( incollare ). Consiste nel moltiplicare ai (coefficiente del filtro) con i campioni x(n-i).

L’idea è quella di calcolare y(n), cioè l’output del sistema al tempo n. Abbiamo bisogno di sapere tutti valori di input. Dobbiamo sapere non solo il campione attuale ma anche quelli precedenti. Un tap è il campione subito precedente a quello all’attuale, il tap comprende anche la ai.

**BUS SPI**

Vantaggio e svantaggio ?

Distinguiamo **slave controllati singolarmente** e **slave connessi a catena**

per i primi:

**Vantaggi:** comunicazione più rapida tra master e singoli slave.

**Svantaggi:** necessità di avere un pin SS per ogni dispositivo slave.

per i secondi:

**Vantaggi:** uso di un unico pin per selezionare i dispositivi.

**Svantaggi:** minore velocità di aggiornamento dei singoli slave, il guasto di un elemento può causare un'interruzione del segnale negli altri dispositivi.

Il vantaggio è che, rispetto all’ i2c, non ha bisogno di un indirizzo fisico del dispositivo ma, *grazie al selec slave (SS), si possono avere tutti gli slave che si vogliono utilizzando pochi bit.*

**Cosa fanno i registri 13, 14 e 15?**

I registri ad uso generale sono 13 (Da R0 a R12)

* **R13** viene usato come stack pointer
* **R14** link register, all’interno di questo registro viene salvato ( dopo l’esecuzione di BL) il contenuto di R15
* **R15** all’interno vi è il program counter

**Cosa serve il link register?**

è usato per salvare l'indirizzo di memoria della prossima istruzione dopo aver terminato l'esecuzione di una funzione eseguendo l'istruzione **BL** o **BLX**.

* **B:** L’istruzione B carica nel PC (R15) l’indirizzo della prima

istruzione della procedura che si desidera eseguire, causando

così un salto nel flusso di esecuzione delle istruzioni. Per

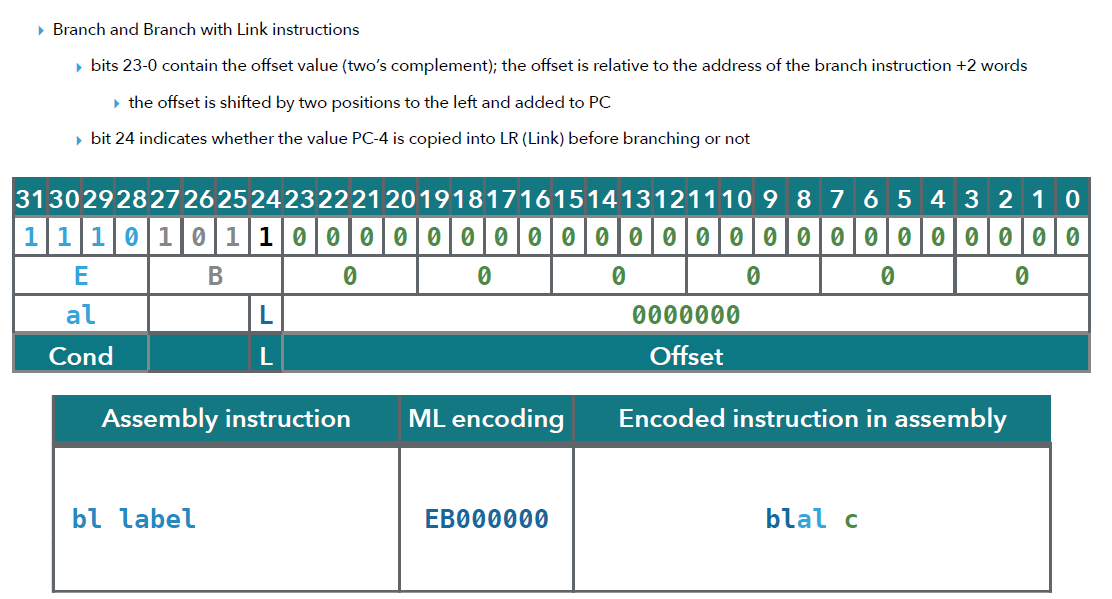
comodità le procedure vengono identificate univocamente con

dei nomi detti label (o etichetta).

* **BL:** simile all’istruzione B, in più però carica nel

registro LR (R14) l’indirizzo di ritorno della procedura, ovvero il

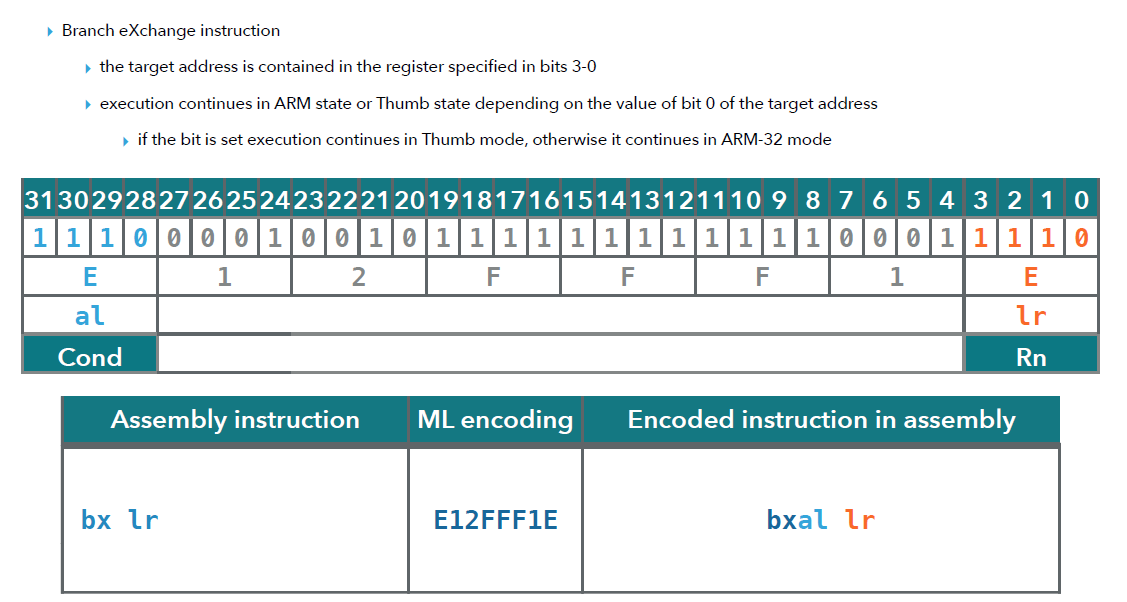
valore del PC nel momento in cui viene eseguita l’istruzione BL.



* **BLX**: the BLX instruction causes a branch to label, or to the address contained in Rm.
* In addition:

The BLX instruction copies the address of the next instruction into LR (R14, the link register).

The BLX instruction can change the instruction set.



**Branch instructions**

Il branch cambia il contenuto del program counter e determina quale istruzione sarà eseguita. L’istruzione a cui saltare viene determinata sommando un valore costante (con segno) al PC, quest’ultimo punta all’istruzione che si trova due word dopo.

L’**offset** è contato in word e quindi l’offset prima di essere sommato al PC viene moltiplicato per 4.

Branch non tiene memoria e branch with link si.

**Chi individua lo stack ascendente e discendente?**

L’istruzione postfissa

**Com'è la pipeline in un processore dsp? Dal punto di vista del programmatore? Che vuol dire pipeline esplicita?**

**Pipelining**

è un macchina semplice **RISC**.

La maggior parte di architetture prima dell’avvento delle RISC consentivano di piazzare dati a qualsiasi indirizzo. Le **CISC** hanno una logica più intricata.

Questo è un modello di cache, prende dati dalla RAM e li elabora qui dentro.

Nei sistemi embedded si hanno due memorie separate, una per i dati e una per le istruzioni, senza memorie cache. Solitamente la memoria delle istruzioni non ha necessità di essere scritta più volte. Le barre grigie sono dei registri, alcuni dei quali potrebbero essere rimossi ma comunque la struttura non cambia e i dati devono fluire sempre da sinistra verso destra.

Si effettuano le seguenti operazioni in sequenza:

**FETCH**

Carico le istruzioni dalla locazione di memoria dell’instruction memory indicata dal PC

C’è anche la logica che gestisce incremento del PC

Il 4 è presente perché questa architettura accetta word di 4 byte e quindi gli indirizzi di memoria saranno tutti multipli di 4.

Il PC attuale va direttamente nell’ instruction memory (istruzione puntata al PC)

Dalle instruction memory escono sempre (in questo caso) 32 bit, e saranno poi decodificati nella fase di decode.

**DECODE**

dei 32 bit due operandi vanno a finire nel register bank e i restanti nel decode.

Dal register bank possono uscire due valori che vanno nello stato di execute

**EXECUTE**

Troviamo ALU che riceve in input i valori del register bank e fa i conti.

Dopo che l’ALU elabora i dati se stiamo usando una macchina CISC (ha 2 registri sorgente di cui uno è quello di destinazione ) il registro di destinazione coincide con quello sorgente e il dato dopo essere stato elaborato viene riscritto nel register bank. Nelle macchine RISC invece ci sono 3 registri (2 solo per sorgente e 1 solo destinazione), uno dei quali è usato per salvare il dato elaborato (quello di destinazione).

NB le macchine RISC non presentano data memory (caratteristica principale)

**MEMORY**

In questo tipo di pipeline riscontriamo il problema della sovrapposizione delle istruzioni **(hazard**) che viene risolto ritardano l’istruzione successiva fino a quando l’istruzione antecedente non ha concluso il writeback.

**CISC**: complex instruction set computer

Un processore con un istruzioni complesse

**RISC**: reduced instruction set computers

**A che serve il bit di start in una trasmissione uart?**

Quando TX vuole trasmettere un frame, invia il bit start (LOW)

SYNC LOGIC in rxr riconosce la transizione HIGH-LOW come condizione di start e avvia il clock. ( vedere sopra )

**Cos'è l'execution token (xt)?**

▸ The word **tick** (‘) provides the eXecution Token (xt) of a word

▸ The xt uniquely identifies a word

**Abbiamo un xt, che operazioni possiamo fare?**

The XT is a cell representing the interpretation semantics of a word, possiamo paragonarlo ad un puntatore in C.

' + .s

1 2 rot execute .

**Cosa può fare uno slave nel bus i2c se non riesce a mantenere la velocità...?**

**Negli slave, tenendo abbassato il master non può inviare i dati. Chi trasmette deve cambiare lo stato del clock in questo caso (meccanismo di strech), qual è il concetto di openDrain?**

Rallenta il clock, tenendo abbassato il clock. Quest'azione produce ??

Open drain connections allows for:

▸ **concurrent operation of more than one I2C master** (if they are multimaster

capable)

▸ **clock stretching** (slaves can slow down communication by holding

down SCL)

**OpenDrain** per le comunicazioni bidirezionali:

* Per trasmette low la logica **attiva PULL-DOWN FET, l**a linea è cortocircuitata a terra
* Per trasmettere high la logica deve **spegnere il PULL-DOWN FET** e usa un resistore di pull-up.

**I primi 4 bit della codifica arm che servono ?**

i bit 31-28 sono i bit di flag:

**NZCV**:

* **N**EGATIVE RESULT FROM ALU FLAG
* **Z**ERO RESULT FROM ALU FLAG
* ALU OPERATION **C**ARRIED OUT
* ALU OPERATION **O**VERFLOWED

4 bit condition

**Differenza tra SPI e I2C ?**

**Full duplex? ( tra SPI e I2C se esiste )**

La prima differenza tra SPI e I2C è che la prima è provvista di un canale full duplex mentre la seconda no.

**SPI:**

è provvista di **MOSI** e **MISO**, il clock proviene dal master.

Le linee di MOSI e MISO vengono utilizzate in base all’occorrenza settando o resettando il bit di read del registro control.

**I2C:**

**Procedura master to slave**:

* Master (trasmettitore) invia condizione di START e indirizza il ricevitore slave
* Master (trasmettitore) invia dati allo slave
* Master (trasmettitore) termina il trasferimento con una condizione di stop.

**Procedura master per ricevere/leggere da slave**

* Master ricevitore invia una condizione di start e indirizza il trasmettitore slave
* Master ricevitore invia il registro da leggere al trasmettitore slave
* Master ricevitore riceve dati dal trasmettitore slave
* Master ricevitore termina il trasferimento con una condizione di STOP.

**Nella subroutine B per richiamare la C, come si comportano i branch?**

Per ritornare da una subroutine è necessario fare il **restore** del **program counter** dal LR.

**MOV pc, lr RETURN instruction**

**bx lr RETURN instruction**

Si usa un’operazione di BRX dove il valore del program counter è memorizzato all’interno del link register, questa subroutine è chiusa da un branch che torna al valore puntato dal link register

Il **branch** cambia il contenuto del program counter e determina quale istruzione sarà eseguita. L’istruzione a cui saltare viene determinata sommando un valore costante (con segno) al PC, quest’ultimo punta all’istruzione che si trova due word dopo.

L’**offset** è contato in word e quindi l’offset prima di essere sommato al PC viene moltiplicato per 4.

Branch non tiene memoria e branch with link si.

Differenza tra lr e pc

utilizzato nelle istruzioni di branch e link

**Parlare dei bus, usart e uart (trasferimenti sincroni e asincroni)**

* Sono connessioni seriali punto a punto (point-to-point. Possono supportare protocolli BUS come **IrDA**, **MODBUS**)
* **UART**: universal asynchronous Receiver/ transmitter
* **USART**: Universal Synchronous/ Asynchronous Receiver /Transmitter
* Sono necessari almeno due segnali digitali:
  + Transmit(Tx) \_ output signal
    - collegato al Rx
  + Receive(Rx)\_input signal
    - Collegato al Tx
* Il range del bit rate va da alcune centinaia ad alcuni milioni per secondo.
* Solitamente UART e USART sono gestite dalle periferiche interne (SOC).
* Esistono due standard:
  + **RS-232**
    - per short range connection
  + **RS-485**
* Le raspberry solitamente sono provviste di due UART:
  + A **PL011** (UART0)
  + A **mini UART** (UART1)

**Differenza tra processori general purpose e DSP ?**

I DSP sono processori costruiti per svolgere compiti specifici in maniera efficiente, i general purpose invece fanno più cose ma non in modo efficiente.

**Cosa usiamo per tornare da una subroutine?**

Un branch exchange al valore del link register

**Differenza tra modalità supervisor e mod interrupt ?**

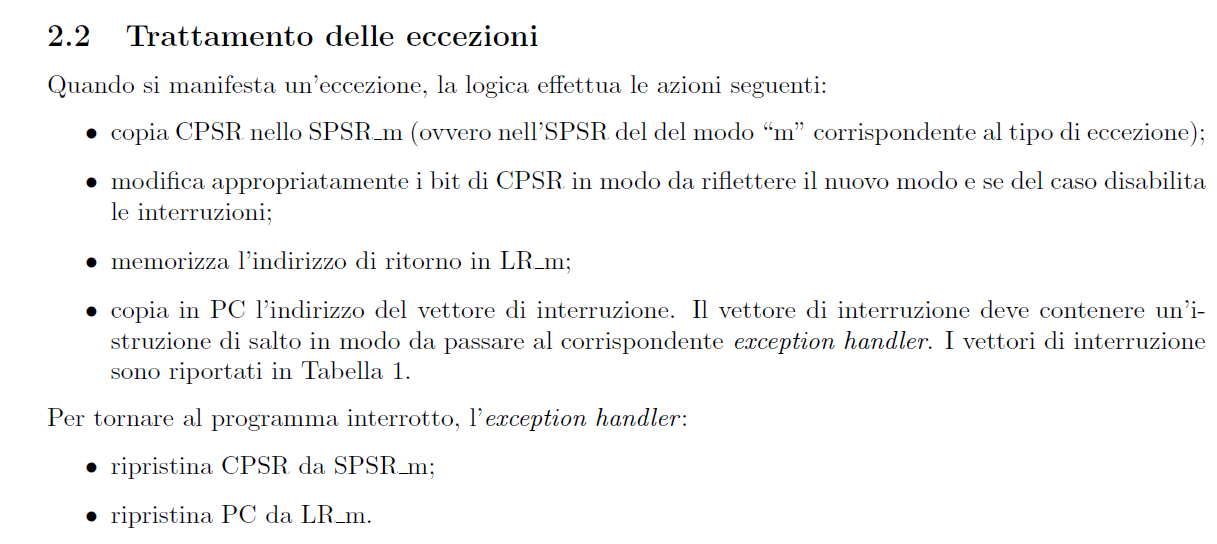
La modalità supervisor (SVC) si attiva quando è eseguita SWI (software interrupt) invece le mod interrupt quando si presenta un’interruzione normale (IRQ) o una fast interrupt (FIQ).

**Modalità del sistema**

* **Supervisor:** entra quando siamo in reset e quando viene eseguita una istruzione di software interrupt (mod privilegiata)
* **FIQ:** entra quando incrementa una fast interrupt ad alta priorità (mod privilegiata)
* **IRQ:** entra quando incrementa una fast interrupt ad priorità normale (mod privilegiata)
* **Abort:** usato per gestire le violazioni di accesso alla memoria (mod privilegiata)
* **Undef:** usato per gestire istruzioni indefinite, quando viene tentata l’esecuzione di un’istruzione non nota (mod privilegiata)
* **System:** modo privilegiato di usare gli stessi registri come user mode (mod privilegiata)
* **User:** modalità sotto la quale molte applicazioni o il sistema operativo girano (mod non privilegiata)

E’ più logico avere la modalità user anziché quella supervisor in un microcontrollore perchè è implementata in maniera più semplice.

ISR o **Handler:**

****

**-1 in complemento a 2, dividendolo per 2 quanto fa?**

**1111 1111 1111 1111 1111 1111 1111 1111 = -1**

**0000 0000 0000 0000 0000 0000 0000 0001 = 1**

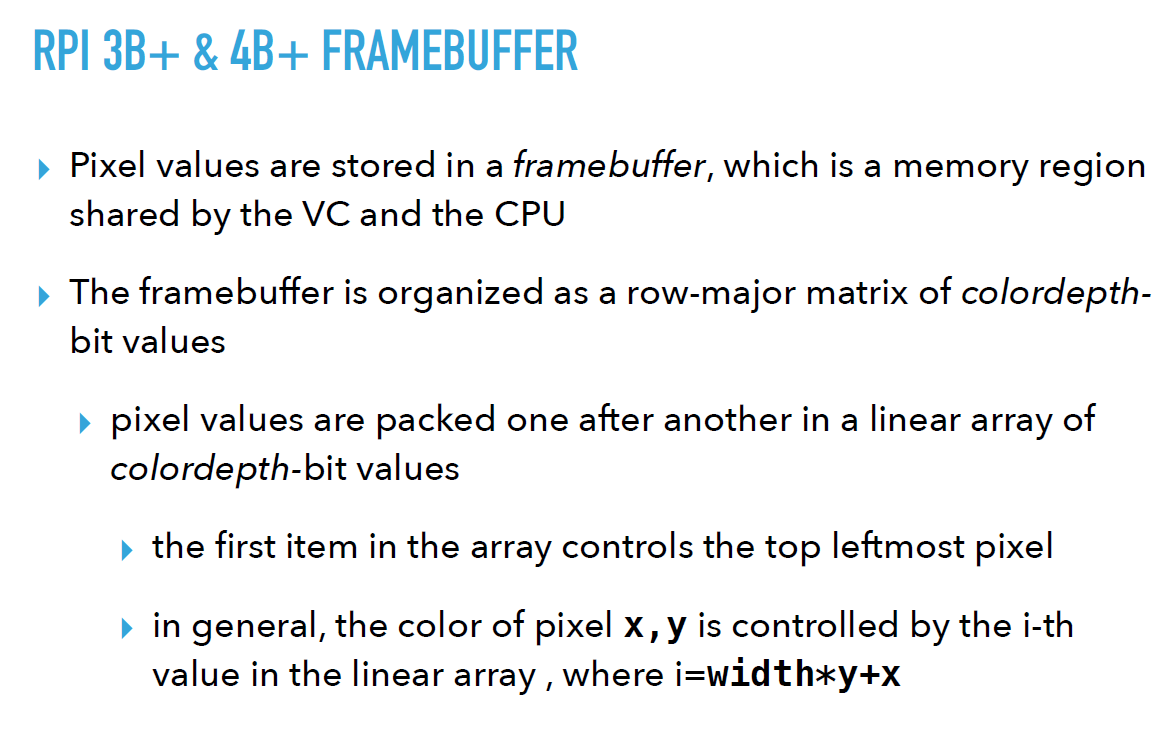
per dividere per 2 dobbiamo **shiftare** il valore a destra, quindi otteniamo un numero positivo ovvero**:**

**0000 0000 0000 0000 0000 0000 0000 0000 = 0**

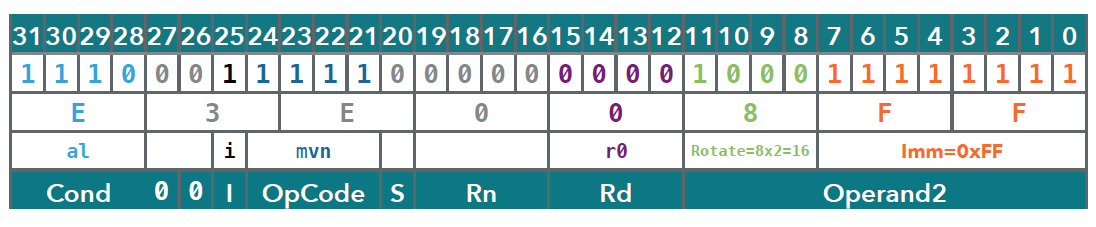
ma noi sappiamo che -½ è un numero negativo, ma noi l’abbiamo ottenuto positivo, come possiamo fare per salvare il bit di segno e ottenere lo stesso valore?

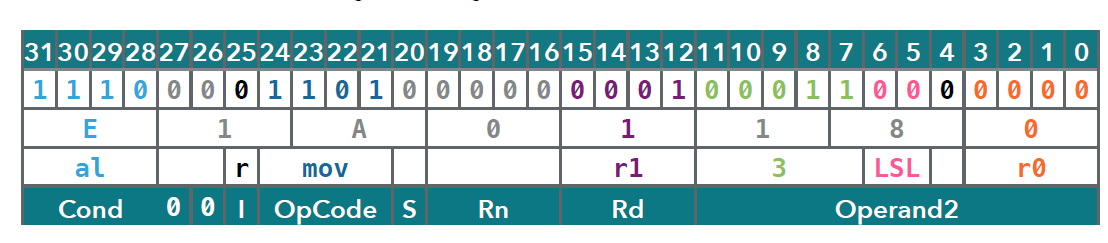
**0000 0000 0000 0000 0000 0000 0000 0001 = 1** anzichè shiftarlo a destra, lo **ruotiamo** verso destra così il primo bit diventa l’ultimo bit che ripristina il segno:

**1000 0000 0000 0000 0000 0000 0000 0000 = -0**

****

**Data processing instruction encoding**

****

****

**Se volessi caricare il valore 0x1ff in un registro come posso fare?**

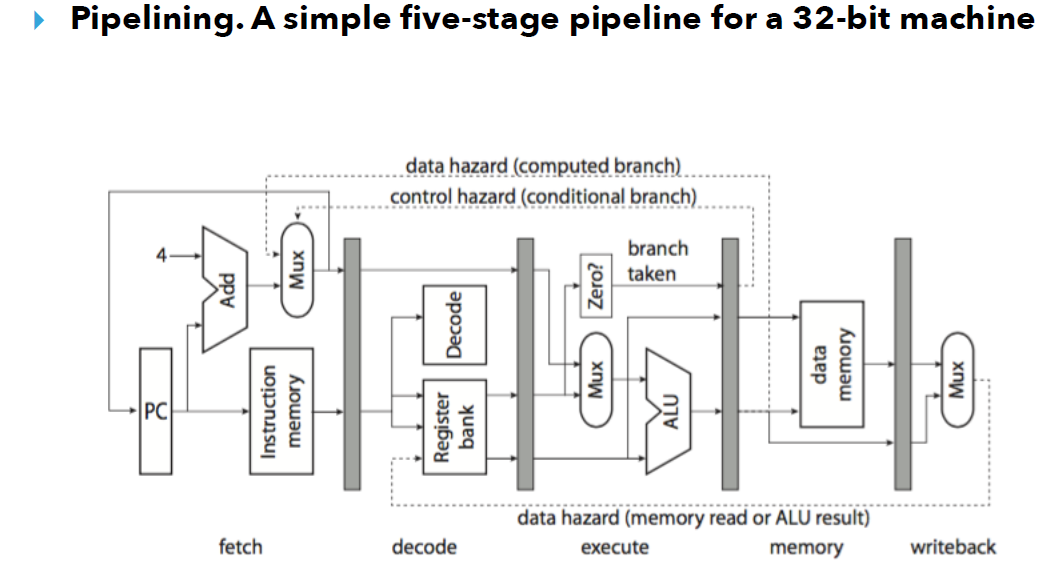
Dato che è un valore a 9 bit non posso usare l’operazione di MOV perchè come secondo operando ha un immediato di al più 8 bit.

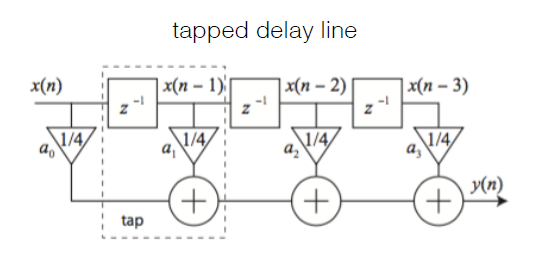
Quindi dobbiamo caricarlo in memoria con un’operazione di di **LOAD:** ovvero mettere il registro di destinazione in cui vogliamo caricarlo e il base register (ci vuole un uguale davanti all’immediato).

**LDR r0, =0x1ff**

registro base pc e mettere l’offset opportuno (dove l’offset è di 8 byte se l’effettiva istruzione si trova al 16° byte)

questo perchè per la struttura della pipeline il pc non punta all’istruzione che sta eseguendo ma a quella che deve essere fetchata.

****

****

**Come si dividono i registri di stack e d’istruzione?**

**Come modifico una parola (tramite un xt)?**

**Quanto possono essere grandi gli intervalli dei registri?**

**Se voglio memorizzare R4 come faccio a capire se è stato memorizzato correttamente?**

**Bit rate framing?**