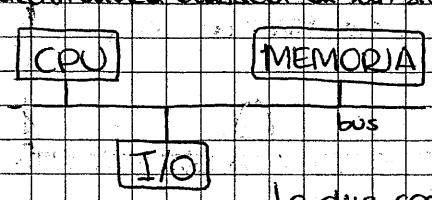
(E1)

ARCHITETTURA CLASSICA DI UN CALCOLATORE

29 FEBBRAIO 2016

L'architettura classica di un sistema di elaborazione consiste in un PROCESSORE, una MEMORIA, che contiene i dati, e delle PERIFERICHE DI INGRESSO/USCITA.



1.a) la **MEMORIA** è un oggetto che sa fare solo due cose ed una alla volta (per di più)

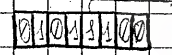
Le due cose che sa fare sono: RISPONDERE a richieste di LETTURA e di SCRITTURA; ciascuna richiesta porta con sé un indirizzo.
La MEMORIA può essere vista come un insieme (una sequenza) di byte, a volte organizzati in unità più grandi (parole, doppie parole, quadruple parole, ...). ARCHITETTURA INTEL
Ciascun byte ha un INDIRIZZO. Quello che può fare la memoria è:
- OPERAZIONE DI LETTURA: dal bus, qualcuno chiede cosa è contenuto all'indirizzo "tot" e la memoria risponde; oppure
- OPERAZIONE DI SCRITTURA: "a questo indirizzo voglio che ci sia scritta questa informazione".

LA MEMORIA NON SA FARE NIENT'ALTRO!
INOLTRE, NON PUÒ FARE QUESTE DUE COSE CONTEMPORANEAMENTE: mentre sta facendo una cosa non può fare l'altra.

Gli INDIRIZZI sono IL NUMERO D'ORDINE DI UNA CELLA, non sono scritti in memoria. Inoltre, quando dobbiamo andare ad un certo indirizzo, NON dobbiamo pensare al verbo "SCORRERE": non c'è niente dentro la memoria che "scorre" qualcosa alla ricerca dell'indirizzo!
Da reti logiche abbiamo visto che nella memoria non c'è niente che "scorre".

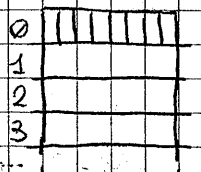
1.b) Si dice che la memoria è **ad accesso casuale** cioè QUALUNQUE OPERAZIONE DI LETTURA O SCRITTURA RICHIEDE UN TEMPO CHE, IN APPROSSIMAZIONE, NON DIPENDE DA QUELLO CHE È STATO FATTO PRIMA E NON DIPENDE DALL'INDIRIZZO.
Quindi, in approssimazione, la memoria impiega lo stesso tempo a rispondere.

2) LA MEMORIA NON PUÒ CONTENERE "NIENTE".



→ In ognuno di questi byte ci sono 8 bit e questi 8 bit sono degli OGGETTI FISICI presenti nella memoria (ad esempio possono essere dei condensatori che saranno carichi o scarichi, oppure possono essere un circuito con un paio di transistor: flip-flop).
Questi dispositivi stanno in uno stato o nell'altro; o sono 0 oppure sono 1 sempre.
Quindi non è possibile che ci sia "niente": ci saranno degli zeri o degli uni, anche quando l'accendiamo, magari casuali.

Dobbiamo immaginare la memoria come una lunghissima sequenza di byte, ogni byte è 8 bit (che sono sempre nello stato 0 oppure 1, ciascuno di essi).



Ogni byte ha un indirizzo (che indicheremo da una parte).

3) DA NESSUNA PARTE È SCRITTO COSA SIGNIFICA QUELLO CHE C'È DENTRO LA MEMORIA. La memoria conterrà sempre dei bit (0 o 1) che, di per sé, non significano niente: non c'è niente che mi permette di capire se questo byte è un numero, oppure un'istruzione, una stringa...
Guardando semplicemente la memoria non si sa!
È il programmatore che sa quale funzione deve avere quel byte in base all'uso che ne fa nel suo programma (sarà un intero se decido che è un intero, altrimenti potrà essere una stringa oppure un istruzione).
Di per sé il contenuto del byte non significa niente, il suo significato dipende dall'uso che se ne fa e questo significato può anche CAMBIARE DURANTE L'ESECUZIONE DEL PROGRAMMA (ad esempio: una volta lo uso come intero, un altra come istruzione...).
Nei linguaggi di alto livello questa cosa non sarebbe possibile, mentre in linguaggio macchina non c'è "nessuno" che controlla il tipo di dato.
Il contenuto del byte NON HA TIPO: il tipo è dato dall'uso che ne faccio e questo "tipo"

può anche cambiare nel corso del programma.

⇒ la memoria fa solo LETTURE e SCRITTURE ad ACCESSO CASUALE (con tempi approssimativamente uguali)
la memoria contiene SEMPRE qualcosa e questo qualcosa NON HA TIPO.

Il **PROCESSORE** è un oggetto "stupidissimo", è un oggetto che ciclicamente:

preleva un'istruzione
↓
la decodifica e
↓
la esegue

Il processore compie questo ciclo all'infinito, fin quando non viene spento.
Non sa fare NIENT'ALTRO.



Gli INTERRUPT comportano qualche modifica, ma molto piccola.
Inoltre, le istruzioni che riesce ad eseguire sono molte, ma sono un NUMERO FINITO.
Ognuna di queste istruzioni è molto SEMPLICE.
Il processore NON FA NIENTE di sua iniziativa, se non gli viene ordinato da un'istruzione.

Anche il processore, come la memoria, non può non contenere "niente" (non può non fare "niente"): se la macchina è accesa, il processore (tranne quando viene messo a riposo, sempre a seguito dell'esecuzione di un'istruzione HLT) fa sempre quel ciclo.

DIFFERENZA TRA HARDWARE E SOFTWARE

Nell'architettura di un calcolatore, una delle cose più importanti è SAPERE "CHI" fa "COSA".
Visto che molte cose si possono fare sia in hardware che via software, la domanda se una cosa viene fatta dall'HW oppure dal SW potrebbe essere malposta; però, se decido una regola di valutazione, in ogni calcolatore è ben definito cosa viene fatto dal software e cosa dall'hardware.

SE PER FARE UNA CERTA COSA CI VUOLE PIÙ DI UN'ISTRUZIONE, vuol dire che quella cosa è FATTA IN SOFTWARE.

PERCHÈ PROPRIO "PIÙ DI UN'ISTRUZIONE"?
Il processore è hardware, ma non fa niente che non sia stato ordinato dal software, quindi tutto in qualche modo è fatto dal SW e tutto in qualche modo è fatto dall'HW (in quanto, di per sé, il software è composto da ordini che poi l'hw esegue).
Quindi tutto è fatto sia dall'hardware che dal software.
Proviamo a vedere la somma tra due interi: ADD %AL, %AH.
È chiaro che devo dire al processore "ADD" e poi dargli due registri; questo è software che dà ordine all'hardware di fare questa cosa.
In questo caso, il programmatore ha scritto questa istruzione ma potrebbe, poi, non saperla eseguire. In caso di errore del risultato dovremmo andare a cercare chi ha progettato il processore in quanto non ci possono essere errori logici nell'istruzione scritta dal programmatore.
⇒ in questo caso, è FATTO IN HARDWARE.

Se, però, le istruzioni sono più di una, il problema potrebbe essere dato da un errore nel software.

Se pensiamo che una cosa sia fatta in SW allora dovremmo anche sapere come si programma (come si scrive quella cosa).

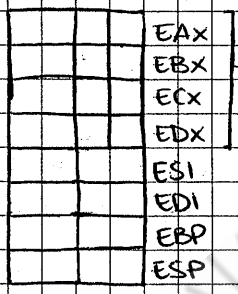
(CE2) Il **BUS** serve per gli ordini di lettura e scrittura.
Questo implica che TUTTA l'esecuzione del mio programma, per quanto complicata sia, vista dal bus, si riduce ad una sequenza di letture e scritture (con indirizzi).

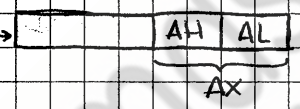
Visto che sul bus ci sono solo operazioni di lettura e scrittura, anche le **PERIFERICHE DI I/O** rispondono solo ad operazioni di lettura e scrittura.
Comunque, la parte di I/O un po' varia in quanto ci sono complicatissime e diversissime periferiche collegate al sistema di elaborazione.
Tutto quello che fanno le periferiche viene tradotto da alcune INTERFACCE contenenti dei REGISTRI.
es. Se do l'ordine ad una stampante e questa vuole "dirmi" che è finita la carta, metterà un bit nell'apposito registro riducendo, così, le operazioni da svolgere a letture e scritture.

ARCHITETTURA A 64 bit

Per poter programmare in LINGUAGGIO MACCHINA, dobbiamo sapere quali sono i REGISTRI, quali sono le ISTRUZIONI e com'è ORGANIZZATA LA MEMORIA.

I **REGISTRI**, nell'architettura a 32 bit erano:





Questi 8 registri vengono chiamati REGISTRI GENERALI e sono di 32 bit.

Nei primi 4 registri (EAX, EBX, ECX, EDX) è possibile accedere ai 16 bit meno significativi (AX, BX, CX, DX) e questi 16 bit sono, a loro volta, divisi in 2 byte (AH/AL, BH/BL, CH/CL, DH/DL) dove la 'H' indica la "parte alta" e la 'L' la parte meno significativa.

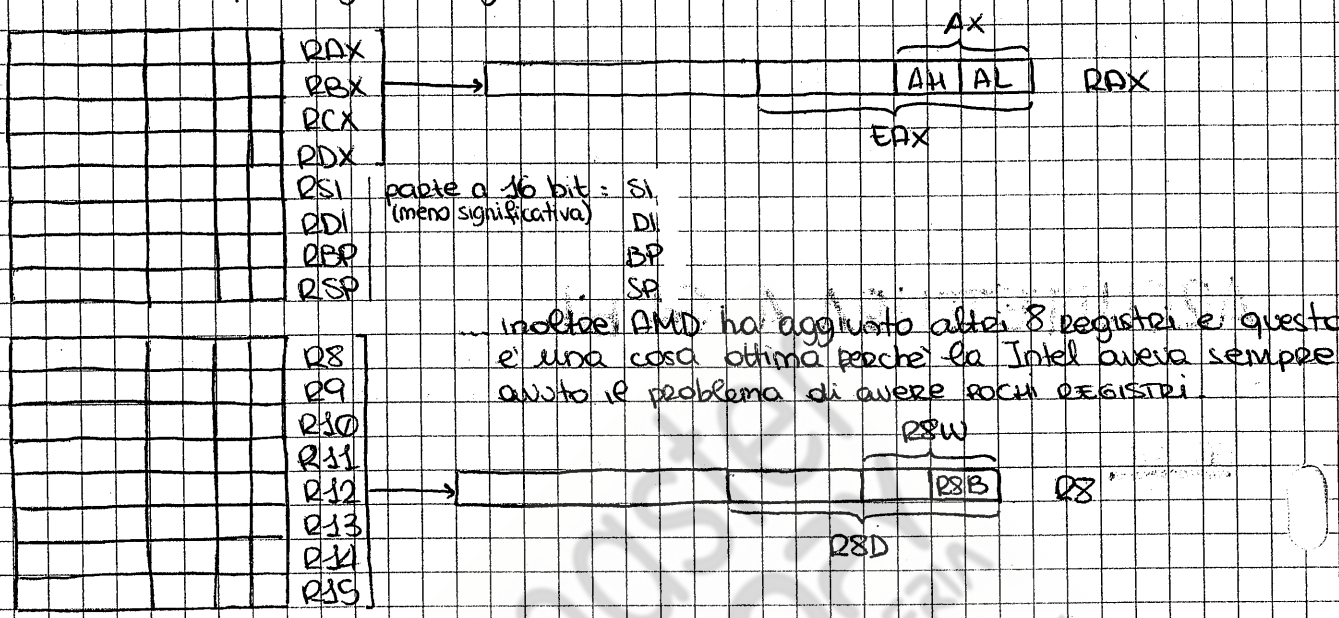
Sono detti REGISTRI GENERALI, ma questo nome tradisce un po'.
I primi quattro registri sembrano in ordine, ma è solo un caso.
- EAX è il registro Accumulatore
- EBX è il registro Base
- ECX il registro Contatore
- EDX è il registro per Dati (una specie di estensione di EAX)

Gli 8 registri **NON SONO INTERCAMBIABILI** in quanto alcuni registri sono "speciali" cioè alcune istruzioni li trattano in modo particolare:
- ESP: le funzioni PUSH e POP per la gestione della pila usano questo (e solo questo) registro.
- EAX: in alcune architetture (antiche) questo era l'unico registro, quindi la moltiplicazione tra naturali utilizza questo registro.
- ECX: viene utilizzato dall'istruzione LOOP come registro contatore, cioè decrementa il registro da sola, però deve essere per forza ECX.
- ESI ed EDI solitamente vengono utilizzati con le stringhe.

Questa architettura è abbastanza "brutta", ma rimase vittima del suo stesso successo: dall'architettura a 16 bit fatta da Intel, venne scritto talmente tanto software che si è dovuto passare a 32 bit.
Le "mutazioni" hanno portato a mantenere tutta una serie di IRREGOLARITÀ (con l'aggiunta della memoria virtuale ed ora il passaggio a 64 bit).
In realtà, la Intel aveva provato a ripartire da zero con un'architettura

che si chiamava IA64, ma non ha avuto successo.
La nuova architettura a 64 bit è stata messa sul mercato da AMD (concorrente della Intel) che ha esteso la vecchia architettura a 32 bit.
Per motivi di compatibilità, troveremo delle irregolarità anche nell'architettura a 64 bit di AMD.

La AMD ha ripreso gli 8 registri a 32 bit e li ha estesi a 64 bit...



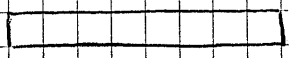
...inoltre AMD ha aggiunto altri 8 registri e questa è una cosa ottima perché la Intel aveva sempre avuto il problema di avere pochi registri.

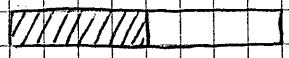
In questi nuovi registri è possibile accedere agli 8 bit meno significativi (R8B, byte), ai 16 bit meno significativi (R8W, Word) ed ai 32 bit meno significativi (R8D, Double).
In questa nuova architettura, i 64 bit sarebbero la "parola" ma, per "storia", la chiameremo PAROLA QUADRUPLA e la indicheremo con Q.

I registri ad 8 bit AH, BH, CH, e DH ci sono sempre per compatibilità, ma è meglio non considerarli e fare finta che non ci siano.
In particolare, se in un'istruzione uno dei nuovi byte (ad esempio R8B), non posso usare contemporaneamente nessuno dei AH, BH, CH, DH.

IN NESSUN CASO è possibile accedere alla parte che resta (i bit più significativi).

Anche gli altri registri sono stati estesi a 64 bit, in particolare l'Instruction Pointer (che viene ad assumere un nome un po' macabro)



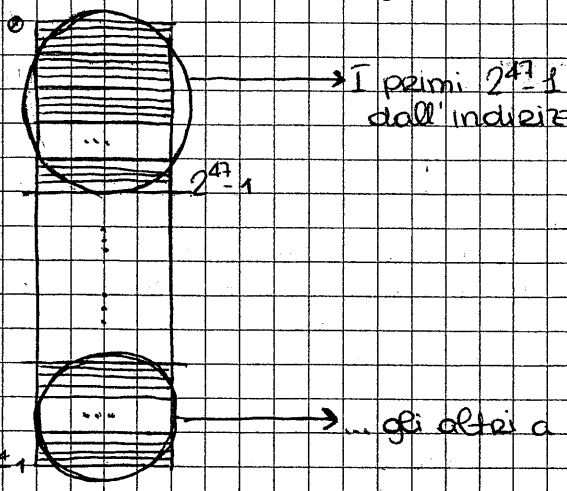


ed anche il registro dei flag è stato esteso a 64 bit, ma della parte più significativa ce ne possiamo anche disinteressare in quanto non sono stati aggiunti nuovi flag (per ora).

Quindi, questi sono i registri che un programmatore deve conoscere.
I registri sono la MEMORIA PIÙ VELOCE a cui si può accedere.

Ora vediamo come è stata organizzata la **MEMORIA**.
Passare a 64 bit serve, soprattutto, ad avere una memoria più GRANDE in termini di indirizzi disponibili.
Anche se, teoricamente, abbiamo a disposizione byte con indirizzi che vanno da 0 a $2^{64}-1$ (una cosa "sterminata"!), nelle specifiche c'è scritto che è possibile usarne anche meno infatti nelle memorie uscite fino ad oggi si utilizzano effettivamente 2^{48} locazioni.

(CE3) $2^{48} \simeq 246.000$ miliardi di byte.
Verrebbe da pensare che vengano utilizzati i primi 2^{48} byte, ma non è così!



Quindi, dei 2^{48} byte disponibili, metà si prendono a partire dall'alto e metà a partire dal basso in quanto tipicamente, la parte protetta viene inserita nella parte bassa.

Ogni volta che si utilizza un indirizzo, l'architettura controlla che questo sia in forma canonica, che appartenga, cioè, alla parte alta oppure alla parte bassa.
Come si può fare questa verifica?
Prendiamo, ad esempio, l'indirizzo $2^{47}-1$, che caratteristiche avrà?
Avrà i 47 bit meno significativi ad 1 e gli altri tutti a 0 00...00,11...1..1
quindi tutti gli indirizzi precedenti (a maggior ragione) avranno i **16** bit più significativi a zero.
Gli altri, invece, avranno i bit più significativi ad 1.
Quindi chi controlla l'architettura, si baserà su questo criterio per vedere se l'indirizzo è in forma canonica.

Ci sono alcune IRREGOLARITÀ:
1) se prendo l'indirizzo della prima cella (tutti 0) e ci sottraggo 1, ottengo un indirizzo con tutti 1 e quindi $2^{64}-1$ e viceversa (se aggiungo 1 all'indirizzo $2^{64}-1$ ottengo l'indirizzo 0).
2) La cosa più "noiosa" è che la AMD non ha raddoppiato la grandezza degli operandi delle istruzioni (teorica per alcune).
Inoltre, non è stata raddoppiata la dimensione degli OFFSET.

Le istruzioni hanno la forma: CODICE OPERATIVO SORGENTE, DESTINAZIONE (standard gcc)
La tipica operazione ha 2 operandi ed una destinazione.
Sorgente e destinazione, si possono trovare con i seguenti metodi di indirizzamento:
- REGISTRO (%RAX): l'operando è contenuto in un registro.
- MEMORIA: letture/scritture in/da memoria.
- DIRETTO ($operando_immediato): direttamente nell'istruzione, è una costante.
Quando l'indirizzo è in memoria bisogna dire, quanto meno, l'<u>indirizzo del byte</u> <u>partenza</u> da cui iniziare a prelevare il dato e la <u>grandezza di tale dato</u> (b = byte, w = word, l = doppia parola, q = quadrupla parola).

Nella sintassi gcc,
La <u>dimensione</u> si specifica con un carattere (b, w, l, q) messo in coda al codice operativo dell'istruzione (ad esempio: movq) ed è NECESSARIO solo quando l'assemblatore NON è in grado di capirlo in nessun altro modo.
Spesso l'assemblatore è in grado di capirlo da solo, ad esempio: se uno degli operandi è un registro, dal suo nome è possibile intuire la dimensione degli operandi visto che solitamente le istruzioni utilizzano operandi della stessa dimensione
Quando, invece, un operando è immediato e l'altro in memoria, allora non è