

A cosa serve lo stack? Esso permette funzionalità come:

- ESTENSIONE PROCEDURALE
- INTERRUPT
- TRAP

Inoltre c'è una struttura di dati di tipo dinamico (può inserire e rimuovere dati nel tempo)

Esso fa riferimento ai registri della CPU e il modo di indirizzamento è di AUTOINCREMENTO *

Per inserire dati nello stack → usa l'autoincremento (= push *)

$$R6 \leftarrow R6 - 2 = RAR[R6]$$

(di -2 perdi con -1 in qualsiasi
de un bit d'oltro della stessa cella)

d'istruzione di MOVE R7, R6 ← R6 - 2 ; RAR[R6]

si traduce in un'operazione di push
nello stack

Lo valore da puntare 1

ultima cella della RAR; seguito l'overflow, lo faccio partire dal valore zero:

$R6 \leftarrow 0$, $R6 - 2$ cerca overflow (intervisita segno)

risultato
"salvataggio
nello stack
dei valori presenti
nei registri"

MOVE R6, R6 ← R6 - 2 ; RAR[R6]

rimane nella
RAR ma non c'è
più visibile a livello
dello stack

crea una copia in memoria del valore, crea la funzione di POP
(torno indietro di un valore e lo riposo dello stack riportandolo
nei registri della CPU)

MOVE RAR[R6], R6 ← R6 + 2, R6
MOVE RAR[R6], R6 ← R6 + 2, R7
J STACK

POP R7

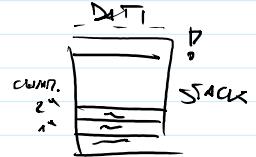
L'estensione procedurale è un meccanismo di intercettazione, con elemento
e salvataggio dei dati nella CPU e nello stack, per poi lanciare l'esecuzione
di un programma diverso (potendo conservare i registri della CPU).
Terminato tale programma, si torna al precedente, recuperando i dati
nello stack.

es.

main () {

} I. sort (...) → il main =^o cangela il suo PC non eseguendo per un po'
 L'istruzione nel programma → registrata nel registro PC
 (program counter)

posso avviare una seconda chiamata dentro sort.



Ogni chiamata occupa una parte di STACK

! il limite e' lo scatto con l'area di memoria dei dati: viene generata una condizione di errore

Es

main () {

$a = 0;$
 $b = a + z;$

PC → C = sort (z);

L → d = s;

;

}

sort () {

es. val.
 152 ————— |
 }

→ prima di passare a sort(), salvo il valore di PC
 (registro che conserva il punto in cui sono arrivato nel main,
 una specie di indice) nello STACK:

MOVE $R7, R6 \leftarrow R6 - 2; R4[R6]$

prendo la cella
 > qui è quella
 già usata

qui copio R7
 ovvero conservo
 il valore di PC

LIFO

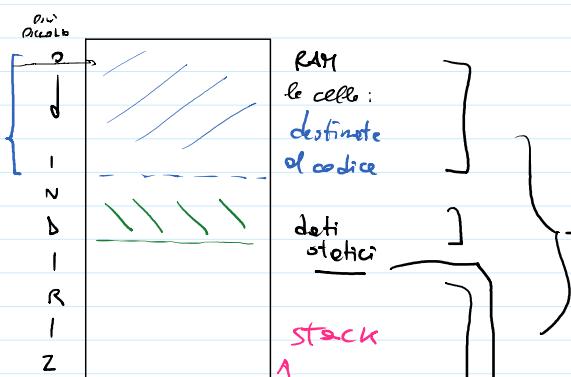
ora posso modificare R4[R6], PC (PC ← 152)

eseguo la funzione sort

ella fine di sort

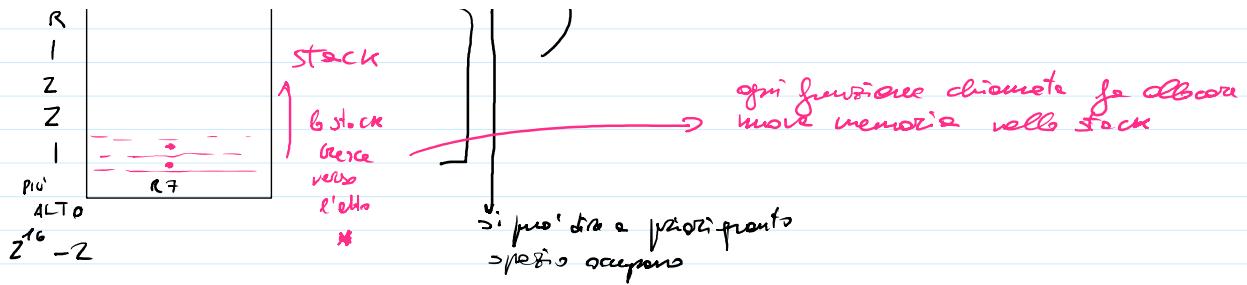
POP R4[R6], R6 ← R6 + 2, R7

cioè ricavo il valore di PC (R7)
 precedente alla funzione sort
 e torno nel main



decodifico in base a grande istruzione vi sotto da eseguire

qui funzione chiamata fa abba

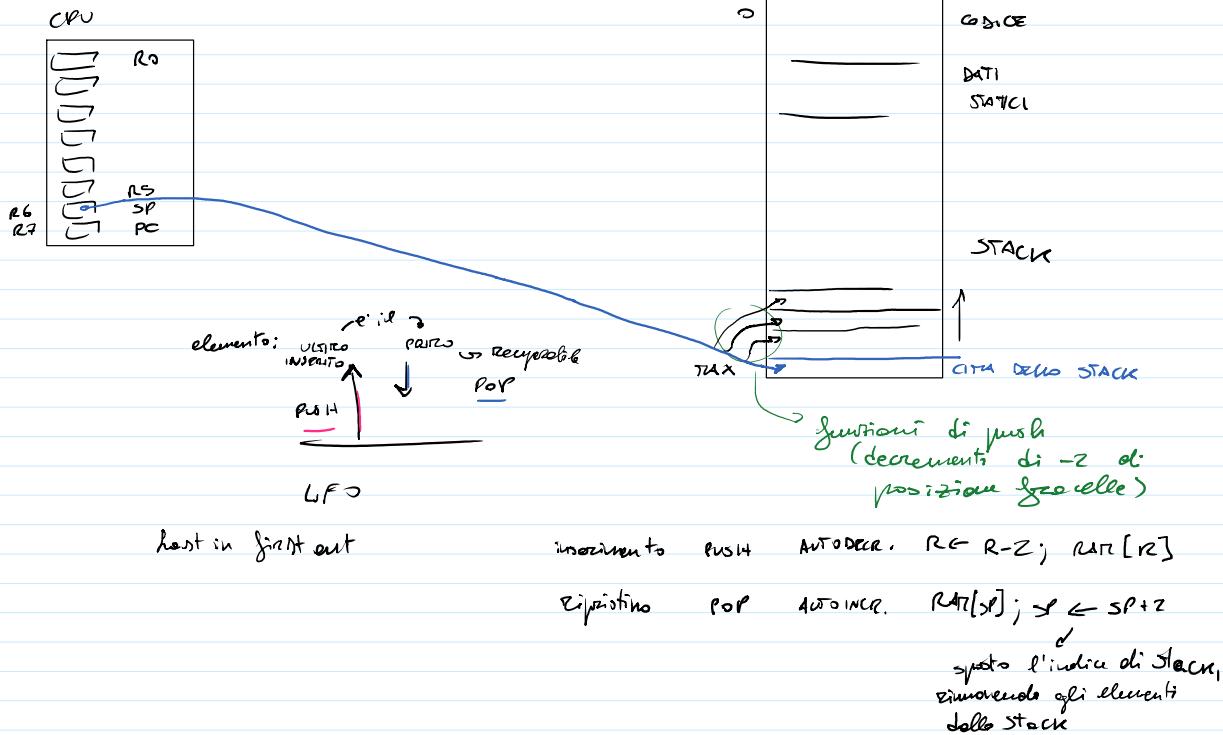


Gli interrupt sono necessari per gestire le interruzioni dal mondo esterno:
 l'esecuzione del programma è autocentrica, una volta avviata l'esecuzione non vi
 è l'idea di interagire con l'esterno. Le interrupt servono a questo: un po'
 come quando uno che suona il campanello, non sapeva cosa vuole finché non vede
 chi c'è, ha quindi un'interruzione delle sue attività e una risposta a tale interruzione.

→ Un programma viene interrotto, l'interruzione segnalata da un dispositivo I/O
 congela il programma e salme i dati nello stack. Si passa all'esecuzione
 di un altro programma: gestore dell'interruzione che indaga sul motivo
 di tale interruzione.

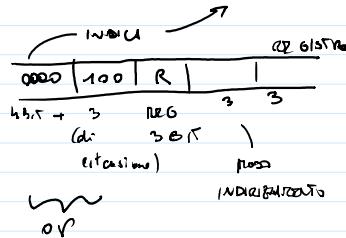
Le TRAP servono per segnalare errori. Ad esempio prende lo stackerino
 alle tante dati (eccede) viene segnalato il problema e il programma viene interrotto.
 Invece la segnalazione di interruzione non viene da un dispositivo I/O ma da
 del programma, bensì del sistema internamente.

Per la segnalazione degli interrupt e i trap si usa il registro 16 nel PDP-11.
 Per volendo si possono usare più registri SP, a cui si può delegare il ruolo
 di gestire interrupt o trap, scegliendo le quantità di spazio da riservare.



- Istruzione di chiamate di procedure (JSR, Jump to subroutine)

Utilizzo con estensione R a 8 bit



Il registro R è implicito nello istruzione, non viene indicato.

- Si effettua una push nello stack

$$SP \leftarrow SP - 2 ; RAR[SP] \leftarrow R$$

contenuto dello stack
posizione in cui punta
l'SP

es. $R = 011$ (3)
corrisponde a:
 $RAR[SP] \leftarrow R3$

dallo stack inserito
il valore contenuto
nella posizione R3

- Si registra in $R \leftarrow PC$, il valore precedente in R è già stato copiato, non rischia di perdere.

(indirizzamento immediato) \rightarrow - Posso quindi cambiare PC, ormai salvato in R:

lo prendo i 16 bit consecutivi
dell'istruzione JSR

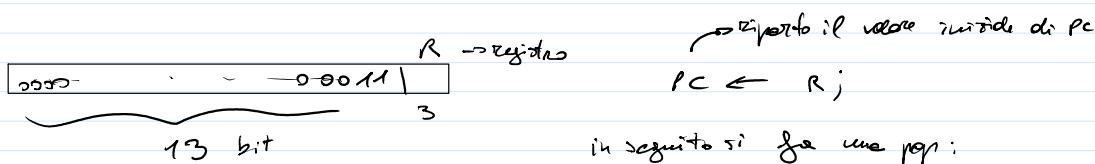
$$PC \leftarrow IND. IMM.$$

N.B. Tale salto di programma è reversibile (posso ripristinare il valore precedente di PC, che è stato salvato in R)

Si può effettuare un'operazione analogo con 2 operazioni IORA (però ho il vantaggio di usare solo un OP e un solo FETTO) (meno cicli di clock, versione appena adatta dell'estensione procedurale)

- Istruzione inversa è il ritorno da subroutine (RTS, return from subroutine)

(è riservata in coda dopo ISR, salta in un programma e poi ne esce)



in seguito si fa una pop:

$R \leftarrow RAM[SP]$; \rightarrow recupero della stessa cella precedente di PC
 $SP \leftarrow SP + 2$; \rightarrow si sposta alla cella successiva
 ("precedente" nello Stack, più bassa)

Ora si lavora con un solo registro dello CPU,
 e si preferisce usare più registri basta formare una push e prima di concludere il programma
 si esegue una pop

che può usare i registri del CPU
 (creando una copia)

Esempio: occupa lo stesso spazio nella RAM & tutto il programma

VAR STATIC

int C = 5;

main () {

int i = 5;

qui arb non sono allocati

i = funz. (7);

PER C PER S
 (ISTRUZIONE SCARICA, INDIRIZZ. DIRETTO, INDIREZ. ROTTO)

\rightarrow viene indicizzata la memoria e viene scritto il S

1° ist. del main 152

"di segn 740

C: 1122

inserita nello stack
 istruzione PUSH e riserva
 spazio per i, mentre per il S uso
 indirizzamento di tipo imm.

tra {} le var obbligate sono
 VARIAIBILI LOCALI (o automatiche)

elle devono di essere
 bloccate automaticamente
 nello stack

CPU (registri generali)

$R_0 \leftarrow 7$

R_1

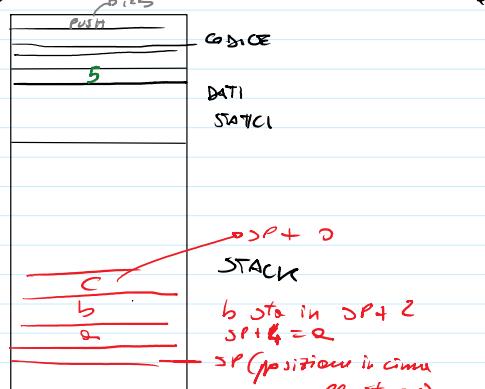
R_2

R_3

R_4

R_5

i numeri di indirizzo sono
 dati dal compilatore secondo
 un algoritmo di elaborazione
 (RAII)



stack

b sta in $SP + 2$

$SP + 4 = Q$

SP (posizione in cima
 allo stack)

C

b

Q

funz. tira
 il valore
 nel registro

(il \rightarrow è comunicato dal main, caricato nel registro)

N.B. Se f. chiamante chiama f. chiamata devono
 seguire quel \rightarrow il registro in cui si
 passano i parametri. In realtà le "convenzioni"
 registrate utilizzate li sceglie il compilatore.

Il limite è il n° di registri

posso scrivere funz. (no parametri diversi) ?

• Sì, anche se ho meno registri da parametri (utilizzo i registri), ma puoi creare dei certi registri che
 possono essere utilizzati

• Realizzo quindi una push nello stack (la funz. ritira
 il parametro con una pop nello stack), lì salvo il
 parametro.

N.B. Nel caso di valori complessi (double, strutture ecc...) ha' una dimensione discendente
 più grande (più indirizzi nella RAM), il tutto viene gestito dal compilatore.

SALVAGGIO E RIPRISTINO DEI REGISTRI

CPU (con registri generali)

$R_0 \leftarrow 7$ R_1 $\xrightarrow{\text{parametro}}$ la funzione si aspetta un certo valore in R_0 e il valore da restituire pu' essere
 salvato su R_0 stesso, dunque si diventa un unico registro di

CPU (can register generators)

$\alpha_2 \leftarrow 7$
 α_1
 α_2
 α_3
 α_4
 α_5

la funzione si aspetta un certo valore in x_0 e il valore da restituire pu' essere
selezionato su x_0 stesso, dunque x_0 diventa un nuovo registro di
memorizzazione per valori e funz.

I Registi "non usati" confermano le istruzioni delle funzioni

La quantità necessaria di registri viene stabilita, es;

uso r_1, r_2 per il funzionamento della funzione (istruzione di funz.)

Perche' un programma non puo' sapere quali registri usa l'altro programma, si supponga che il main li sot tutti, si esegue comunque;

le istruzioni
 di PUSH e POP
 vengono eseguite
 come poste
 dal compilatore
 prima di modificare
 i valori nei registri

PROT R₁
 PUSH R₂
 ;
 ;

l'importante è non cambiare i valori negli altri registri
 (non ne fa fatta una copia!)

POP R₁
 POP R₂

prima dell'esecuzione del programma recuperare i valori
 precedenti in R₁ e R₂
 (l'uscita "risultato" delle funzioni le posso salvare in R₀)

risolvo la quantità
 di memoria che
 viene usata (si
 utilizzano gli stessi
 registri, e i valori che
 estengono sono memorizzati → • allocando sullo stack mi serve il max; 300 byte
 dinamicamente e temporaneamente)
 • allocando in modo statico $50 + 250 + 300 = 600$ byte

Scrivere / ritirare
Come accede RE alle varie locali durante l'esecuzione del programma?
(es. i s nel main)

N.B. la cella non necessariamente e' in linea con lo stesso (non si recupera facilmente col port).

Si voce un modo di indicare come è stato rispettato, rispetto alle sp (unità di rinnovabilità delle tecniche pop)

$\text{CAT}[\mathbf{sp} + \underline{\mathbf{g}}]$

$\text{Ran}[\rightarrow_p]$
celle in einer
ell. struktur

→ tale distanza c'è una costante costante da sommare
e se poi poter accedere al valore nell'indirizzo

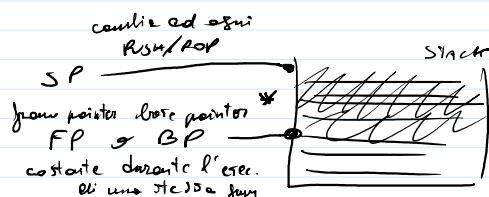
$$2\pi[s\rho + \zeta]$$

(vedi disegno con a, b, c)

Per indicizzare lo stock si usa un registro in più

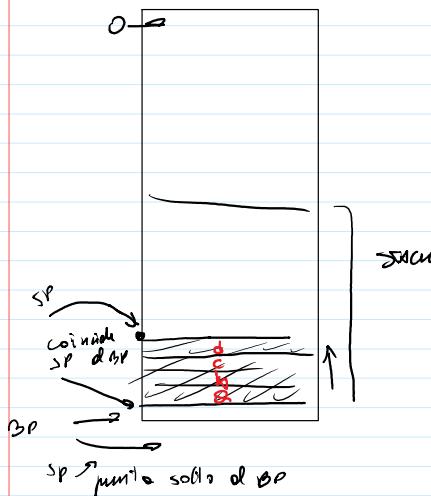
Tale manica semplifica al programmatore il ritrovamento delle zone in cui le verdi locali sono solcate, oltre a un'ottima lettura della strada.

Lo permette di pensare la struttura delle memorie
onda e tempo di esecuzione



col' BR combine
grands to JSR or RTS

Edu.



- all'inizio del main si punta sotto BP

- push; SP coincide con BP

- push con le variabili, SP cresce.

Se voglio recuperare i valori di var., mi muovo rispetto al BP:

$$a \rightarrow BP + 0$$

$$b \rightarrow BP - 1$$

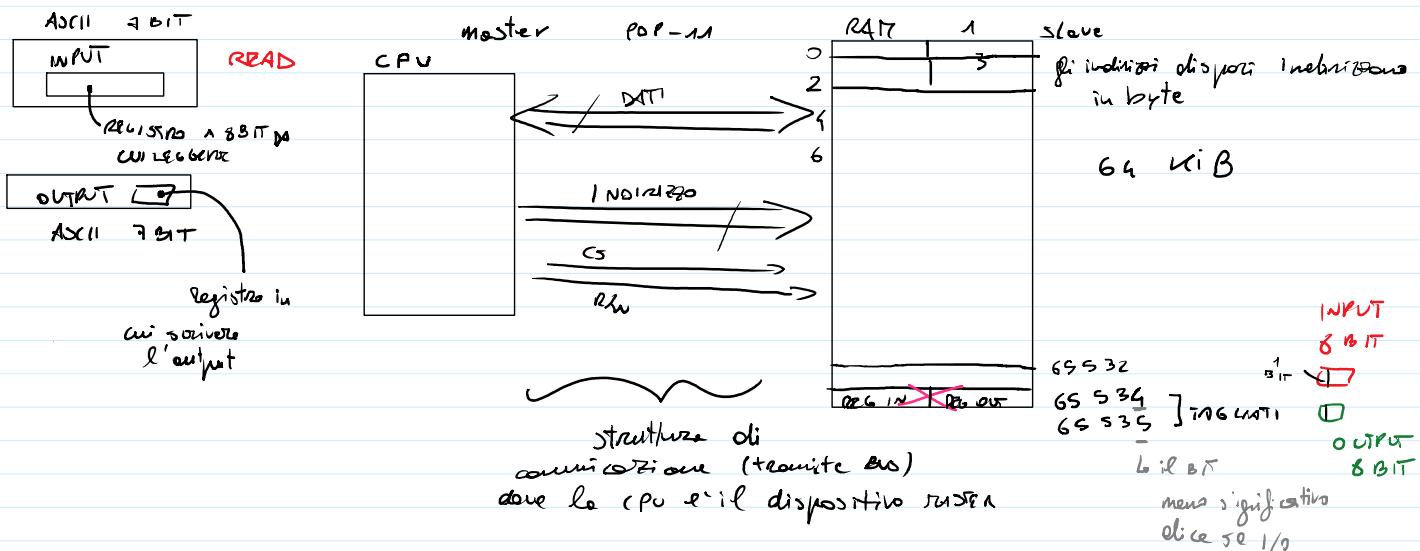
$$c \rightarrow BP - 4$$

$$d \rightarrow BP - 6$$

(mi allontano dal BP)
cioè: delle cime
della stack

Dispositivi I/O - mappatura registri

giovedì 3 dicembre 2020 16:34



Potrei realizzare un dispositivo che restituisce un'uscita dato un certo input.

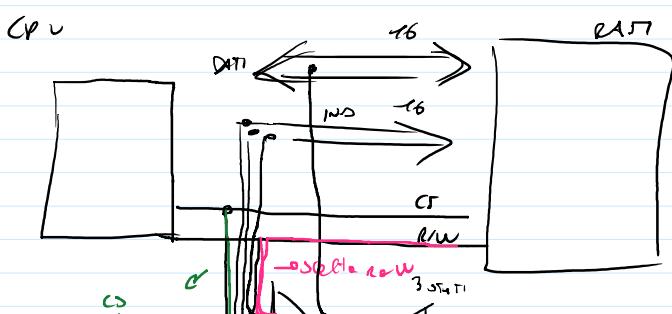
Esempio: INPUT una tastiera i cui caratteri sono codificati secondo il codice ASCII a 7BIT e l'uscita potrebbe essere un display che mostri i caratteri codificati in ASCII a 7BIT

MAPPATURA IN RETRODIAPORTA DEI REGISTRI (Registers Memory Mapped)

- elimina delle celle di memoria della RAM
- posso prendere ora il registro di input e metterlo al posto del Byte 65534 appena eliminato
(i registri rimangono funzionali nei dispositivi di I/O)
- così facendo la RAM puo' accedere ai registri di input e output e la CPU puo' interagire con i dispositivi di I/O senza ulteriori complicazioni, ordinando a leggere delle celle di memoria della RAM.
- le celle di memoria viene letta quando e' fatta l'operazione. Ma bisogna controllare se c'è da e' stato inserito nelle celle un carattere presente in ci' che l'utente ha inserito. Vale a dire se il bit inserito nel registro sia corretto per inneggiatore. Tale informazione "si verifica", come un bit di segno.

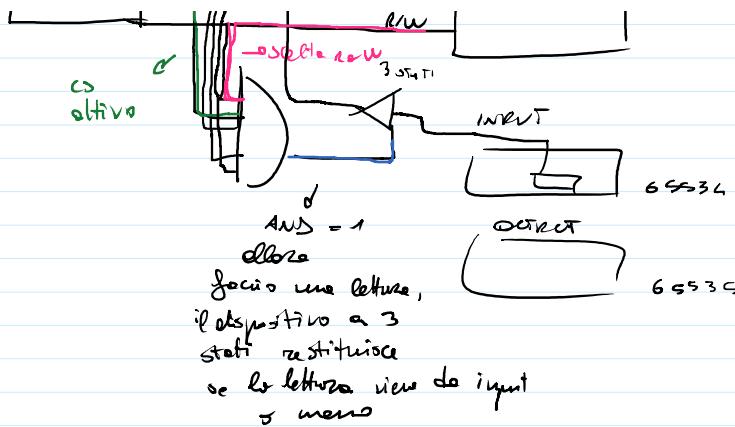
+ (0) carattere valido → i 7bit consecutivi sono le codifiche corrette effettivamente presenti
segno / - (1) nessun carattere (il pulsante del carattere non si è stato premuto)

Quindi c'è più nulla da leggere in questo momento



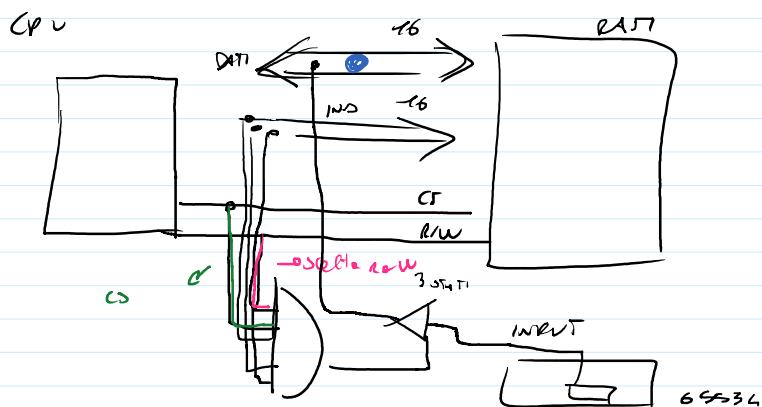
Alla pressione di un tasto ha un interrupt del processore con conseguente risposta (nisi tempi più brevi possibili)

Per lo lettore posso verificare se



Per la lettura posso verificare se esca si rivolge al registro input o meno

Dispositivo di uscita



metto i dati al dispositivo di output

l'ultimo bit permette di gestire il tempo necessario al disp di 0 per portare a termine la scrittura

se troppo lento rispetto all'invio dei caratteri da parte della CPU rischia di avere sovraccarico; deve ritardare la scrittura finché il disp di 0 non è pronto per stampare il carattere successivo.

stamp leggo il
bit più significativo
prima di scrivere

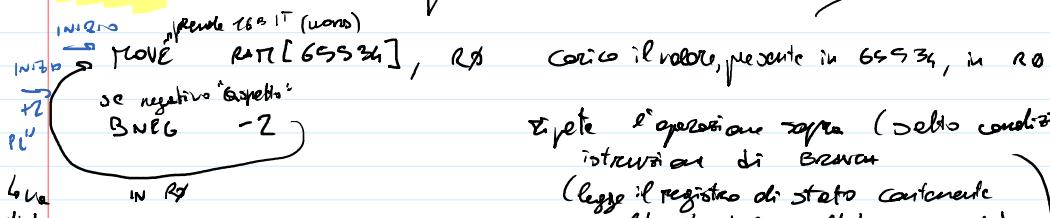
0 + sta amore stampando non pronto
1 - scrittura terminata pronto

dopo che il bit di segno viene ripetuto a 1 dal dispositivo stesso
al termine delle scritture del carattere

seguono estensione delle bit di segno



LETTURA (veloce & non produce dati)



} attesa attiva
(busy waiting)
L'uso cicli di
dado delle
cose continue ad
eseguire l'attesa

RL
L'una
di due
in due

' istruzione di Branch
(legge il registro di stato contenente
l'esito del risultato precedente)

- decine delle
CPU che continuano
e eseguire l'elenco

gestisco io
la memoria
RAM

SALV ASSOLUTO
JUMP 15321
salto e un certo indirizzo
 $PC \leftarrow 15321$

while se non
un'interruzione della
gestione delle
memorie e le o.
voglio saltare indietro
di n posizioni)

Salvo NENTRO
Branch + DELTA
(ha un incremento pos -> neg)
 $PC \leftarrow PC + \Delta$
↓
nel
caso
es., $PC \leftarrow PC - 2$

Scrittura (di un carattere)

move B (prendendo 8 bit) es.
move B RAM[65535], R1
se pos B pos -2

leggo se il valore nell'output (avverto se il carattere è stato stampato)

dispositivo non pronto, attendo (busy waiting)

move B R2, RAM[65535]

1^o lettura

es. "Ciao", ogni carattere viene presentato

do R2 e scritto, riportato su R1 (con l'informazione se
pronto o no), dispositivo si mette in output
il carattere successivo da R2.

1^o pos. non pronto
2^o pos. pronto

2^o scrittura del carattere
(pos)

3^o al termine scrittura
segno 0 -> 1

Vi sono quindi metodi migliori (el punto
del busy waiting) tali da impegnare
in modo più efficiente la CPU
(es. con un interruttore che permette di
stampare il carattere successivo quando
il corrente è stato stampato)] = INTERRUPT

Mecanismo di interrupt

