

- Come può essere realizzata una PARTE OPERATIVA.

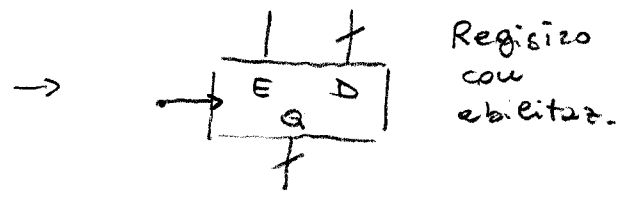
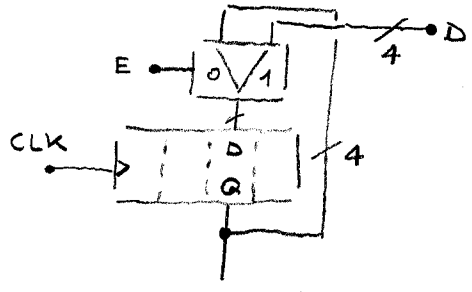
> Il progetto è fortemente influenzato dalle applicazioni per cui è pensato il processore

- Elaborazioni di calcolo di tipo generale
- Elaborazione di segnale
- Rappresentazione dell'informazione, tipi di dati

> Ha una struttura "data path", formata da registri, bus, e blocchi operatori

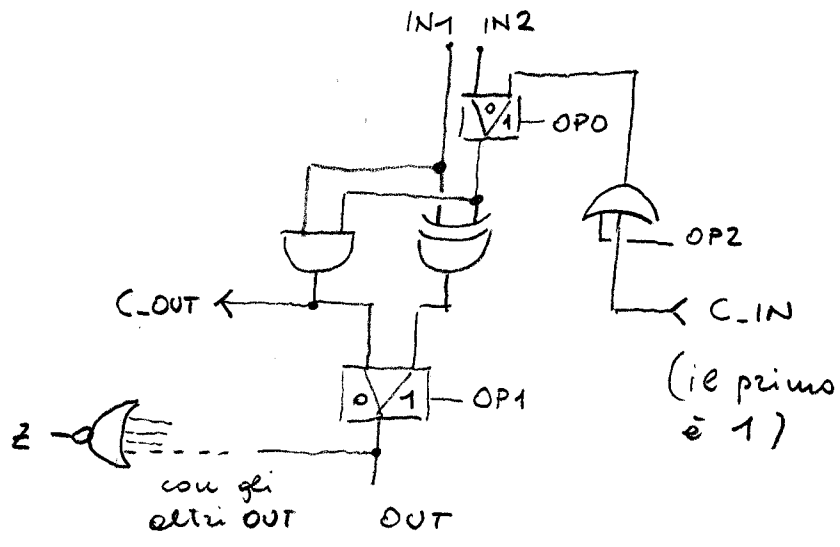
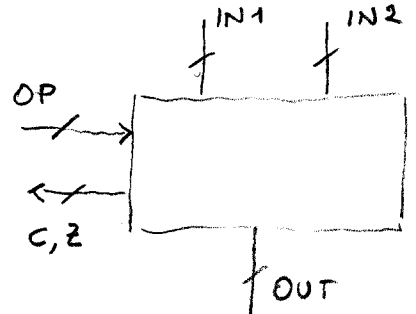
> Proponiamo, a titolo di esempio, una parte operativa con cui si possono realizzare semplici operazioni logiche e l'incremento.

- La dotiamo di 4 registri intercambiabili di uso generale (a 4 bit) con abilitazione "E"
- Diamo la possibilità alla parte di controllo di caricare i registri con un valore "immediato"
- Prevediamo un operatore INC, AND, NOT, XOR con operazioni logiche bit-wise.



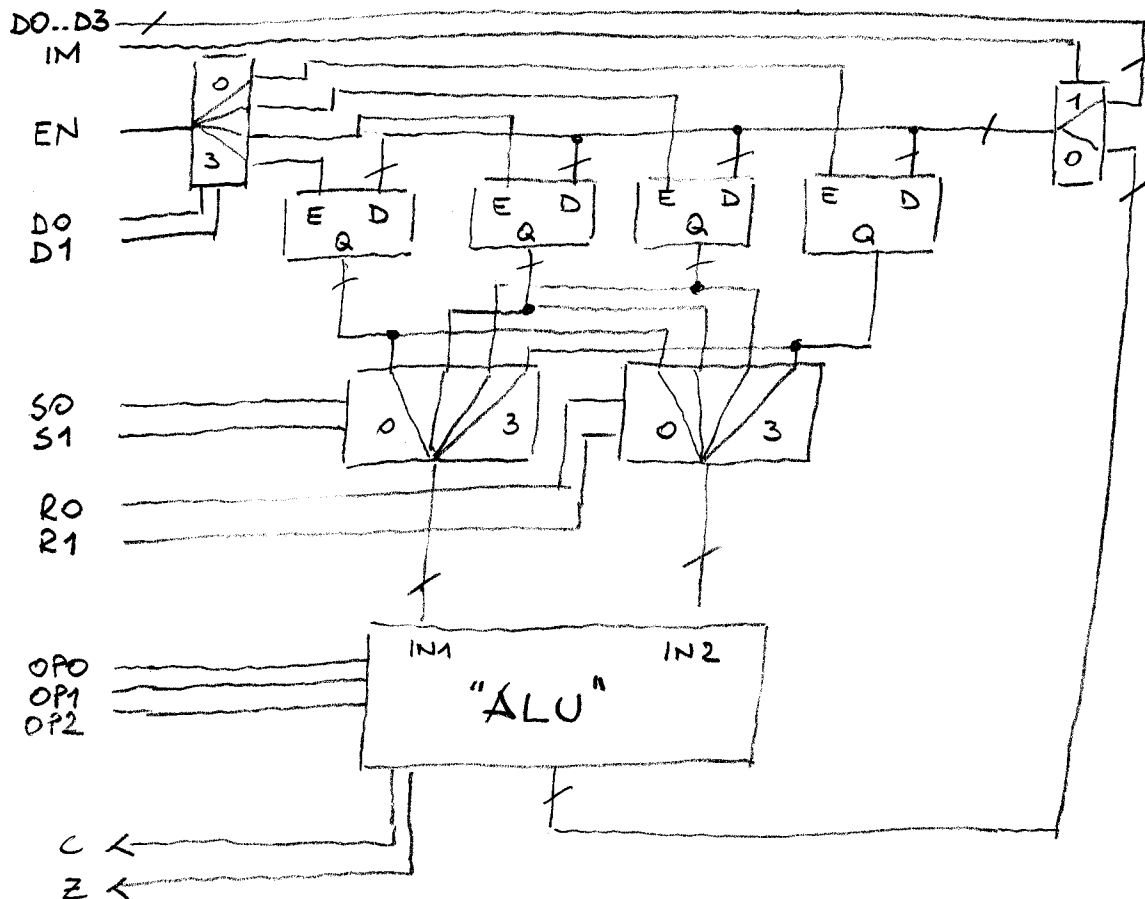
Registro con abilitaz.

Blocco dell'operatore



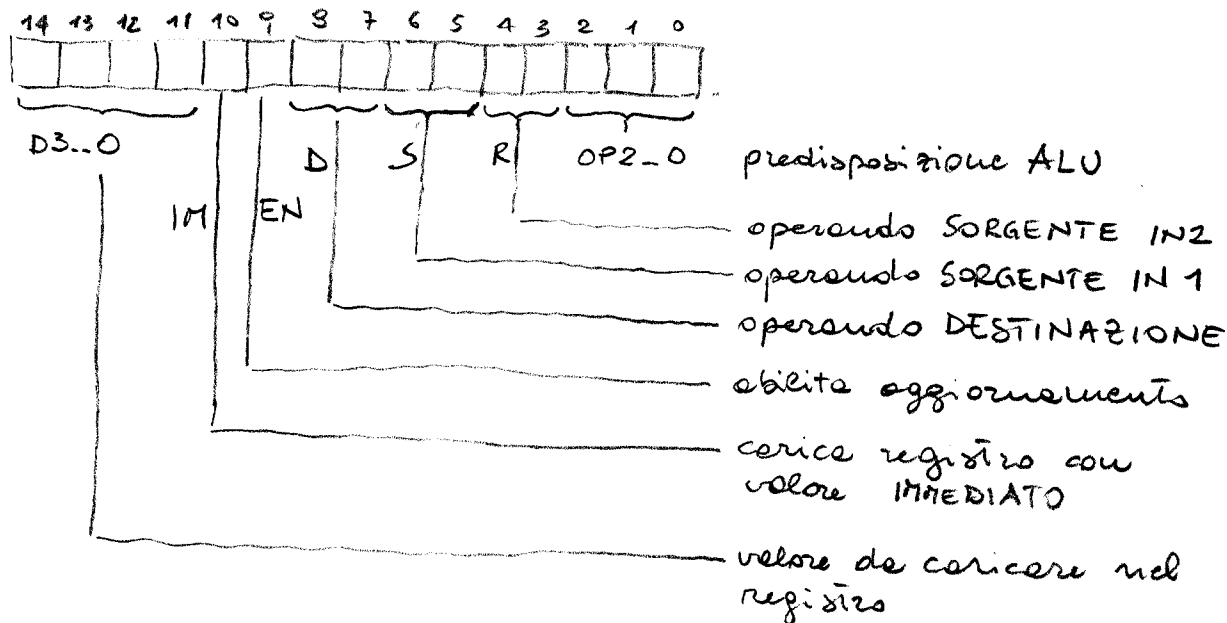
OP2	OP1	OP0	Operazione
0	0	0	AND (IN1, IN2)
0	0	1	-
0	1	1	INC (IN1)
0	1	0	XOR (IN1, IN2)
1	1	0	NOT (IN1)

• Si ha quindi:



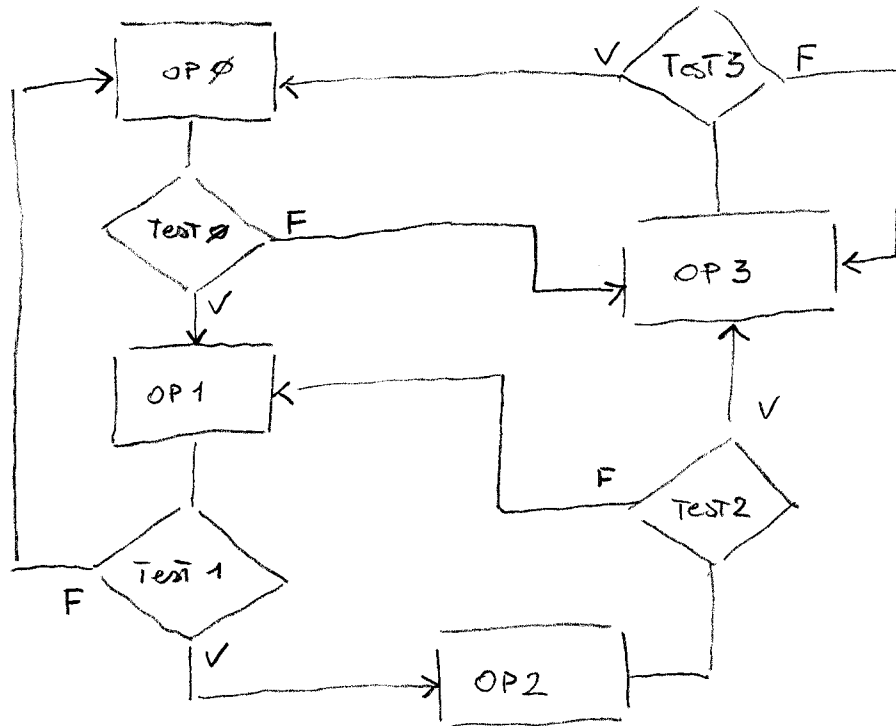
> Questa parte operativa prevede quindi "istruzioni" da 15 b, che predispongono le varie parti per eseguire operazioni richieste

• NON tutte le combinazioni dei 15 b sono significative



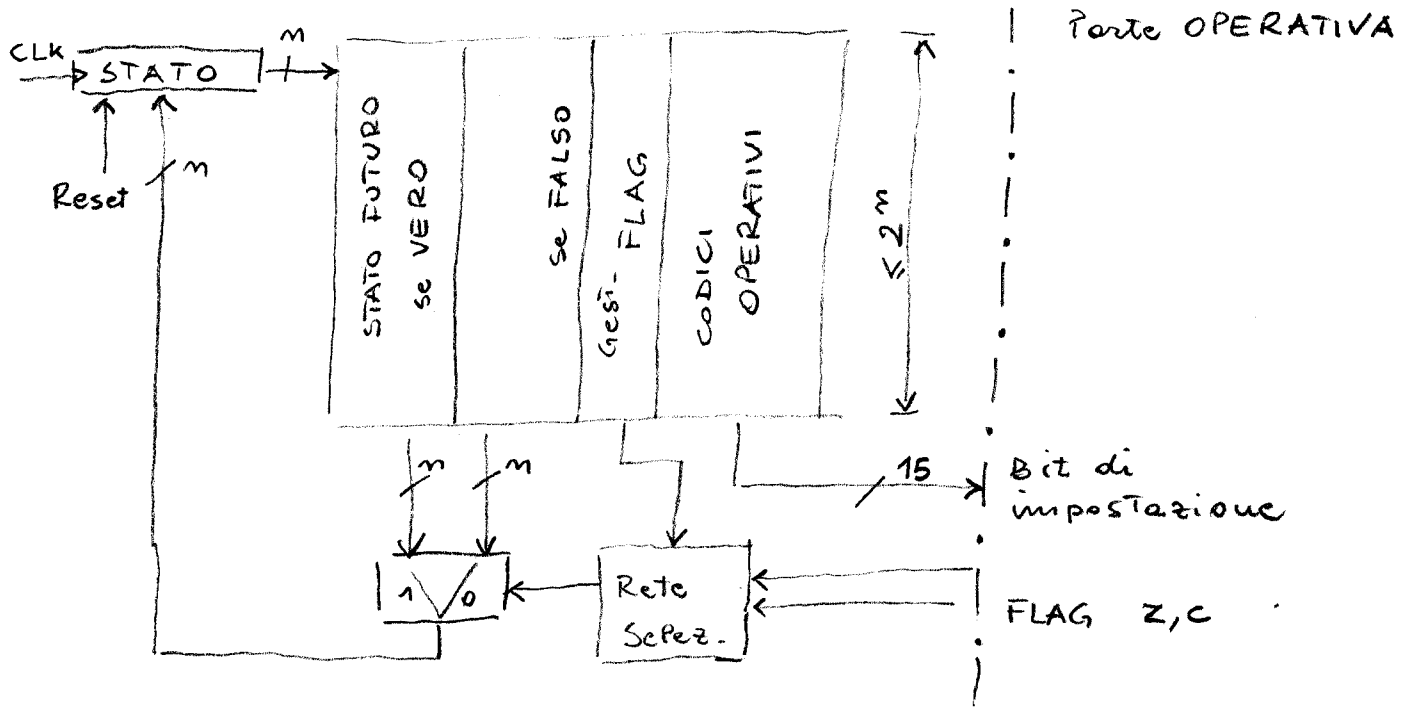
- Possiamo immaginare di realizzare la parte di controllo in modo "strutturato", al fine di abilitarla a eseguire flussi di operazioni MODIFICABILI sulla base delle operazioni esterne

1.10

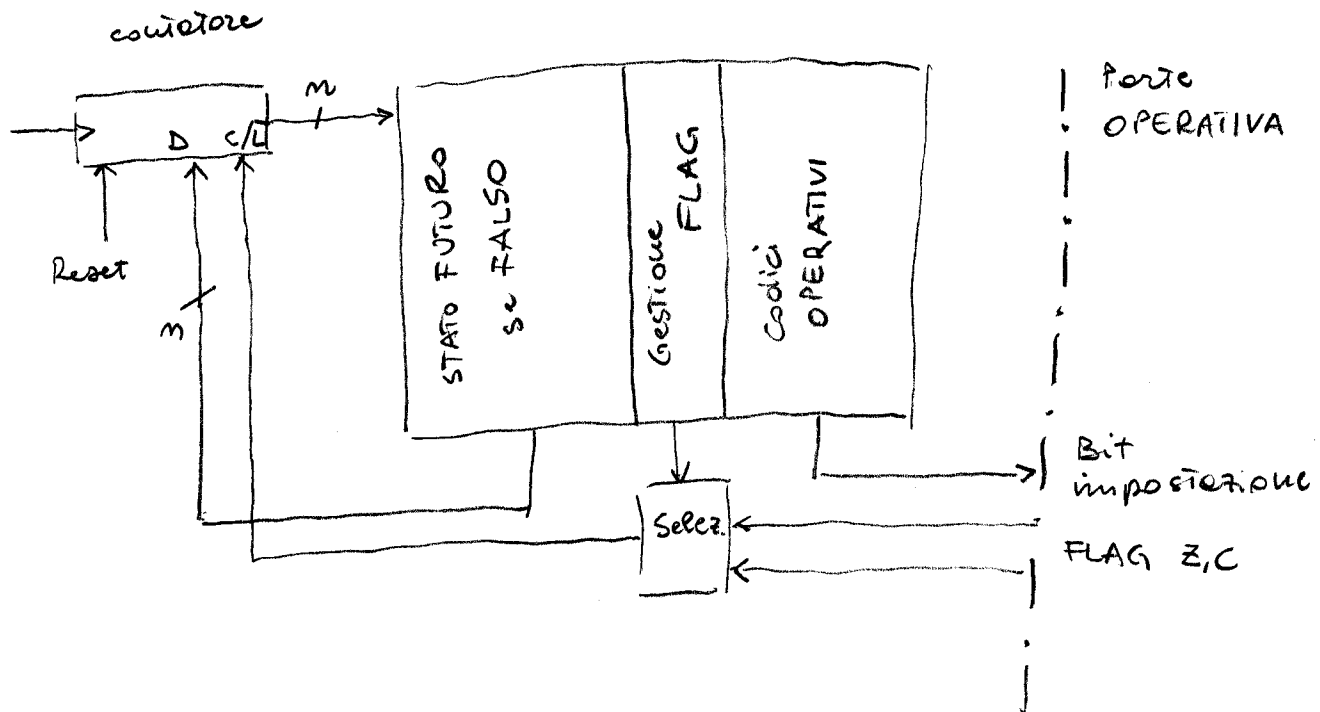


- > Le OP vengono eseguite "imponendo" una certa (15 b) parola con segnali di controllo alla parte operativa
- > I test sono eseguiti su combinazioni logiche di segnali provenienti dalla parte operativa (Z o C)
- > Si usa, perciò un'architettura, basata su MEMORIA, per realizzare questo diagramma che prende il nome di SEQUENZIA TORE.

- Serve un REGISTRO di appoggio per indicare lo STATO corrente ( $n^{\circ} \text{STATI} = 2^{n^{\circ} \text{bit}}$ )
- Serve un meccanismo per decidere lo STATO futuro, impostando il test
- I bit che gestiscono l'OPERAZIONE devono essere inviati alla parte operativa (sono i 15b visti prima)



> Se le operazioni previste nell'esecuzione del programma, vanno eseguite SEQUENZIALMENTE, la struttura può essere significativamente semplificata usando un contatore con CARICAMENTO parallelo



- Possiamo vedere cosa significa programmare quella struttura in modo da realizzare, in RO, un contatore che conta continuamente da 5 a 11 e riparte ogni volta da capo.

> Si tratta di scrivere il contenuto della memoria, in modo che le operazioni siano corrette

• Facciamo riferimento all'architettura con registro

Indirizzo Stato	FUTURO 1	FUTURO 0	FLAG	CODICE OP	Comm
0	1	-	1	5, 1, 1, 1, -, -, -	R1 ← 5
1	2	-	1	11, 1, 1, 2, -, -, -	R2 ← 11
2	3	-	1	-, 0, 1, 0, 1, 1, 0	RO ← R1
3	4	-	1	-, 0, 1, 0, 0, -, 3	INC RO, RO
4	3	2	Z	-, 0, 0, -, 0, 2, 2	XOR RO, R2, -

> Siamo riusciti a ottenere una funzionalità "evoluta" solo cambiando il contenuto della memoria interna del processore, cioè il MICROCODICE.

- Limiti dell'approccio "POPC" proposto

> Le "istruzioni" sono in forma "estesa" - Richiedono grandi quantità di memoria.

> La memoria del  $\mu$ CODICE influenza pesantemente le prestazioni del processore -

• Non può essere troppo grossa

• Deve essere facile da "raggiungere" e modificare

> SOLUZIONE:

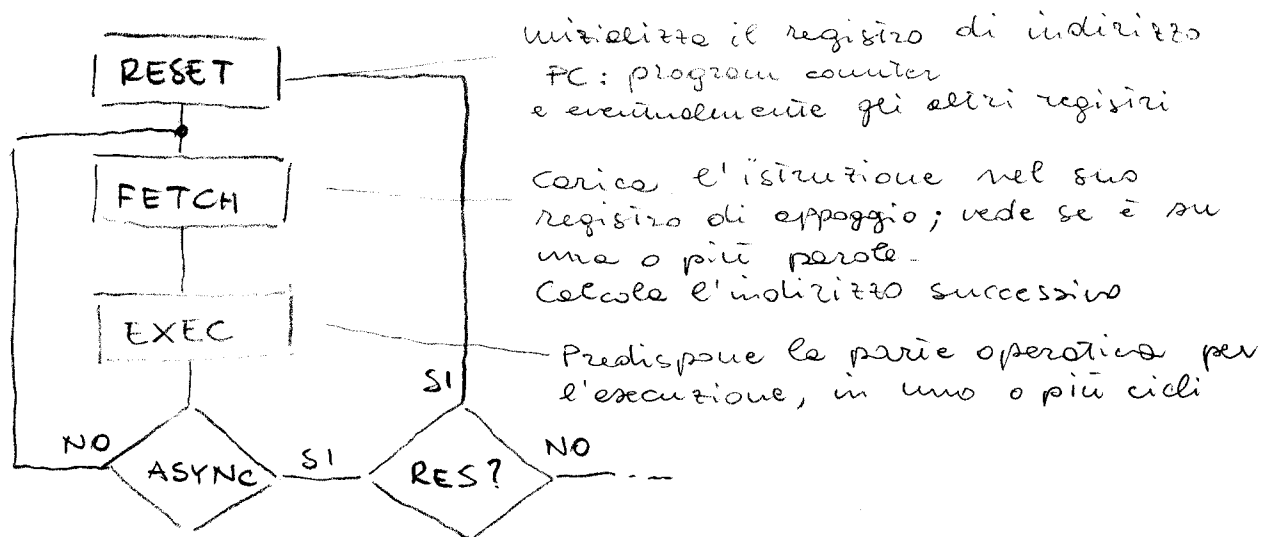
• Usare un secondo livello per codificare le istruzioni in forma COMPATTA

• Usare il  $\mu$ Codice per costruire un INTERPRETE di questo nuovo tipo di istruzioni (livello ASSEMBLER)

- Processore con comandi assembler

L'architettura di questo sistema prevede una parte CONTROLLO specializzata, in grado di dialogare con una memoria esterna tramite un bus

- > Servono alcuni registri di appoggio, per l'indirizzo della memoria e per le istruzioni caricate da eseguire
- > Occorre stabilire un flusso ciclico, che dopo una inizializzazione, prelevi sequenzialmente (ricorrendo le eccezioni) le istruzioni, le esegua.
  - Si può prevedere anche la possibilità di interrompere con segnali esterni (RES, INT, HALT) questo flusso



• Per "scompattare" l'istruzione assembler occorre una rete combinatoria di decodifica, che interpreti i diversi campi che sono stati inseriti nell'istruzione

> Proseguendo nel nostro esempio, possiamo immaginare un processore con il seguente assembler:

```

    INC RD
    AND RD, RS
    XOR RD, RS
    NOT RD
    LD RD, RS
    LDI RD, n
    JP n
    JP c, n
    NOP
  
```

(CODICI "MNEMONICI")

- Possiamo organizzare i comandi su parole (1 o più) di 8 b, eseguendo vari campi



codice OPERATIVO

- indica QUALE operazione
- in CHE MODO sono indicati gli operandi e DOVE trovarli
- specifica SE il comando è su 1 o più parole
- specifica SE occorre fare salti

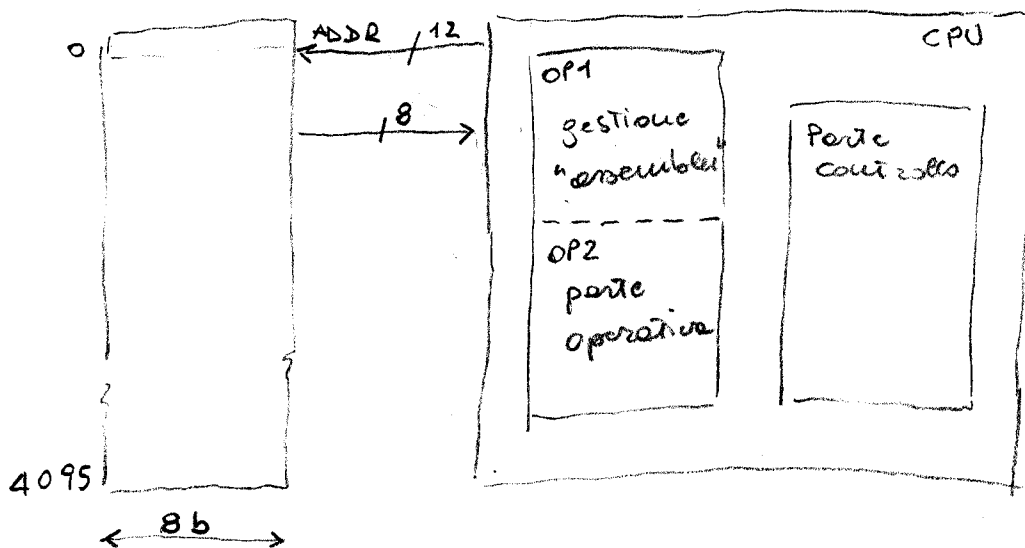
operandi o altre informazioni utili per eseguire l'operazione

- Proviamo a costruire i codici per le istruzioni proposte (codice mnemonico → codice binario)

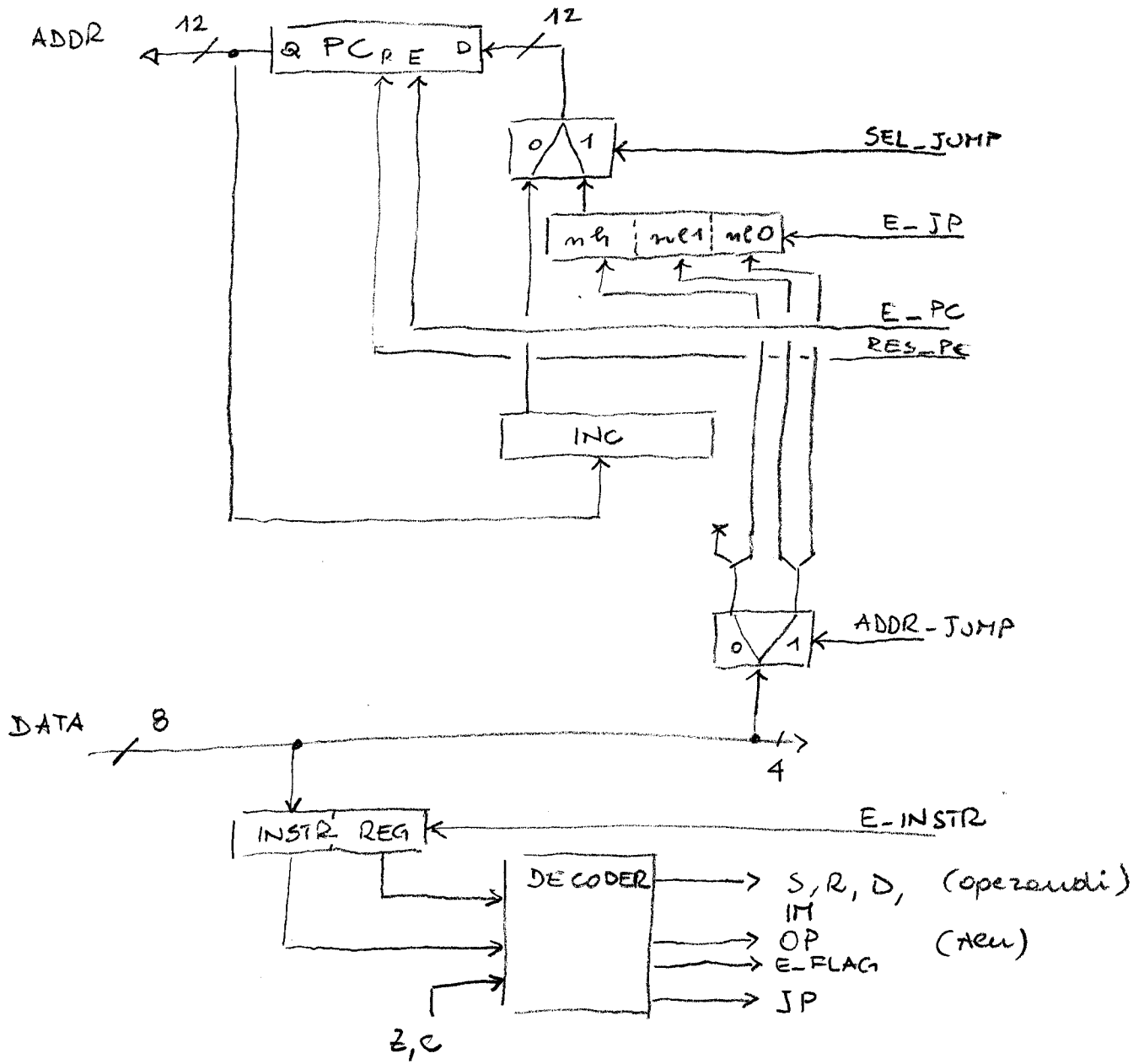
NOP	00000ddd	1	D, S
LD RD, RS	0001D S	1	00 R0 01 R1 10 R2 11 R3
LDI RD, m	10D m	1	
NOT RD	0010D do	1	C
INC RD	0011D 100	1	00 Z 01 NZ 10 C 11 NC
AND RD, RS	0100D S	1	
XOR RD, RS	0101D S	1	
JP C, m	11C mh	ml	2
JP m	0111 mh	ml	2

- Abbiamo fissato in 4096 locazioni lo spazio di memoria e in 12 b la dimensione del registro PC

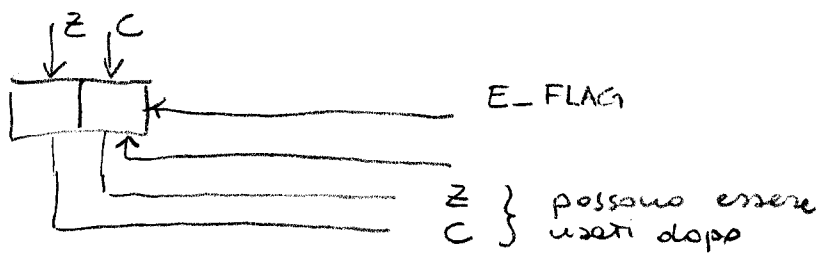
MEMORIA EXT (ROM)



> I nuovi registri possono essere considerati alla stregua di quelli della parte operativa Vediamo di specificarli

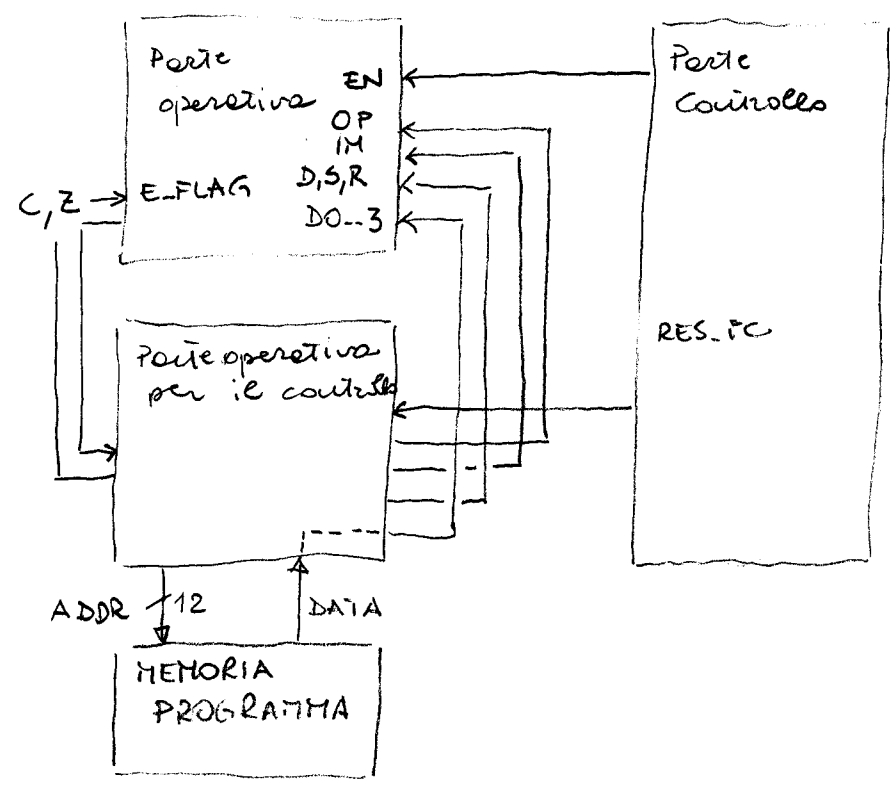


> Nella parte OPERATIVA occorre aggiungere un registro (FLAG) per memorizzare il valore dei bit Z e C dopo le operazioni aritmetiche.



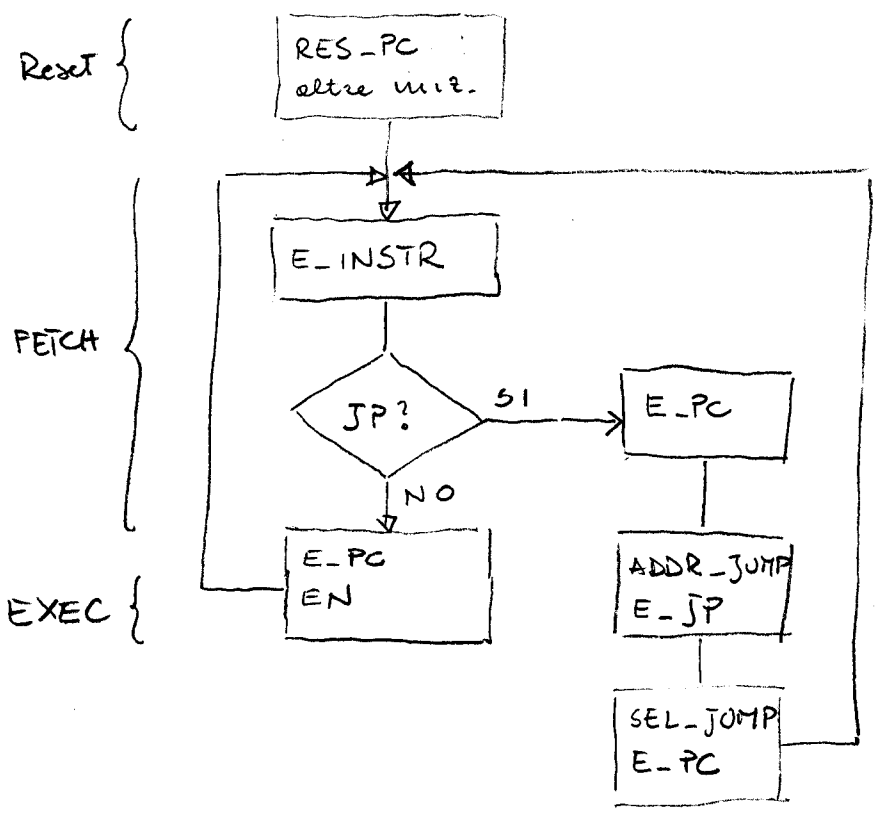


> Ricapitoliamo le nuove configuration per potere progettare il sequenziatore



- Possibile microcodice

- Macchina a 6 stati  
Una istruzione "normale" richiede 2 clock
- Il salto ne richiede 4



- Vediamo il programma assembler che realizza il precedente contatore 1.77

byte	ADDR	ISTR	8 byte
1	loop:	LDI R0, 5	
1	loop1:	LDI R2, 11	
1		INC R0	
1		XOR R2, R0 ; confronto con 11	
2		JP Z, loop	
2		JP loop1	

- Tempo di esecuzione del ciclo 70 cicli di clock

TRACCIA

CLK	ISTR	Stato
0	LDI R0, 5	R0 = 5
2	LDI R2, 11	R2 = 11
4	INC R0	R0 = 6
6	XOR R2, R0	R2 = 11 XOR 6 ≠ 0
8	JP Z, loop	non eseguite
12	JP	eseguite
14	LDI R2, 11	R2 = 11
16	INC R0	R0 = 7
18	XOR R2, R0	R2 = 11 XOR 7 ≠ 0
20	JP Z, loop	
24	JP	
	⋮	
60	JP	
62	LDI R2, 11	
64	INC R0	R0 = 11
66	XOR	
70	JP Z, loop	eseguite

Ripete

- Set di istruzioni di un  $\mu P$

- > Caricamento e spostamento dati  
Registri, memoria, I/O
- > Esecuzione di operazioni  
Logiche, aritmetiche, spostamento di bit, selezione di bit
- > Controllo del flusso di esecuzione  
Salti, sottoprogrammi

- Architetture CISC/RISC

> Set di istruzioni  
COMPLESSO

Maggiore capacità operative  
Istruzioni "lente"

> Set di istruzioni  
RIDOTTO

Pochissime istruzioni veloci  
Individuare le più usate e  
ottimizzarle

- Architetture Harvard/Von Neumann

