1Crisi

PDP8

Compilatore : https://mircot.github.io/html/js_pdp8_app.html

Il PDP8 conta le posizioni in esadecimale, quindi se c'è scritto che inizia da org 200 il comando subito dopo l'org sarà a 200, quel duecento è esadecimale.

La rappresentazione dei 'numeri' utilizzata nel PDP8 è in complemento a 2, quindi, letti in tal modo, il range dei numeri rappresentabili è [-32768,32767]

Se trovo un ADD 310 per esempio, senza che specifichi il tipo, punta alla cella

Se trovo LDA X e X è BUN X I, per trovare il valore da mettere nell' acc monto il seguente codice

X	XXX	XXXXXXXXXXXX
1 se indiretta sennò 0	Cod op.	Org della X

Gli org si contato 209 20A 20B 20C 20D 20E 20F 210 211 …

Quando overflowa il registro E si chiappa un 1

1.Comandi

ADD: Somma tra AC e la cella indirizzata
LDA: (Load Accumulator), carica in AC il contenuto della cella indirizzata
STA: (Store Accumulator), salva nella cella indirizzata il contenuto di AC
BUN:(Branch unconditionally), salto incondizionato alla cella indirizzata
BSA: (Branch and save return address), salva l'indirizzo successivo al suo nella subroutine che richiama, dopo riparte da dopo la subroutine (in teoria salva la "posizione" dell'istruzione successiva al bsa nella cella indicata e trasferisce il flusso del programma alla cella di memoria successiva a quella indicata, ma se ci ritorna il programma non continua lì ma ritorna all'istruzione successiva a BSA)
ISZ: (Increment & skip if 0), incrementa di 1 il contenuto della cella indirizzata, e se uguale a 0, salta l'istruzione successiva
CLA: (Clear Accumulator), azzera AC
CLE: (Clear link), azzera E
CMA: (Complement accumulator), inverte gli 1 e gli 0 (es. 7→-8 NO NEL CASO OVERFLOWA)
CME: (Complement link), complementa logicamente il contenuto di E
CIR: (Circulate right), sposta di 1 i bit verso dx in E-AC, equivale a dividere /2. Stessa cosa della CIL ma dal verso opposto

CIL: (Circulate left), sposta di 1 i bit verso sx in E-AC, equivale a moltiplicare x2. Il primo bit (quello che viene levato) va a finire nel registro E, e in fondo alla stringa su cui ho fatto la CIL viene messo il valore di E precedente all'operazione.
INC: (Increase accumulator), incrementa di 1 il contenuto di AC
SPA: (Skip on Positive AC), Salta l'istruzione successiva se AC>0
SZA: (Skip on Zero AC), salta l'istruzione successiva se AC=0
SNA: (Skip on Negative AC), salta l'istruzione successiva se AC<0
SZE: (Skip on Zero E), salta l'istruzione successiva se E=0
HLT: Arresta il sistema

Comandi indiretti
STA X I : Salva il contenuto dell'accumulatore nella cella il cui indirizzo è specificato nella cella X

LDA X I : Carica nell'accumulatore il contenuto della cella in cui indirizzo vale X

AND X I : Esegue l'AND logico tra l'ac e la cella di memoria puntata da X

ADD X I : Esegue la somma tra l'ac e il contenuto della cella a cui punta X

BUN X I : Salta alla cella puntata da X

BSA X I : Salva l'indirizzo della cella successiva all'interno della cella di memoria puntata da X

ISZ X I : Incrementa il contenuto della cella puntata da X, se il risultato è 0 salta alla cella successiva

2.Overflow dell'AC

Per quanto riguarda un overflow di valori nell'AC, quest'ultimo assume i valori in maniera circolare, nel senso se io ho una valore che supera il 32767, appena gli aggiungo +1 esso riparte a contare da -32768

es:

AC può contenere valori nell'intervallo [-32768, 32767]

Voglio inserire il valore 37903

37903 - 32767 = 5136 5136-1 = 5135 -32768 + 5135 = -27633

Non c’è un controllo rispetto ai valori che vengono caricati, valori che superano il range consentito (-32768, 32767) vengono mappati nel corrispondente valore consentito.

es : X, DEC 65000 = X, DEC -536

3.Istruzioni indirette (I)

L'istruzione `BUN X I` implica un salto indiretto, il che significa che invece di saltare direttamente all'indirizzo `X`, il programma salta all'indirizzo memorizzato nella locazione `X`.

Ciò non vale solamente per BUN ma per qualsiasi comando che abbia la “I”.

4.Come calcolare tick di clock del programma

-Conti il numero di istruzioni (org non conta).

-Lo moltiplichi x4

Credo si calcolino così:

*-Cicli di clock pdp8 = cicli di macchina * 4 = (numero istruzioni + numero stadi - 1) * 4*

5.Come cercare di capire se un programma cicla all’infinito

L'analisi manuale del codice è un primo passo fondamentale. Cerca i seguenti elementi che possono indicare la presenza di cicli infiniti:

- Loop Espliciti: Cerca istruzioni di salto (`JMP`, `JMS`) che possono formare cicli.
- Condizioni di Loop: Verifica le condizioni dei cicli (`ISZ`, `SMA`, `SZA`, `SNL`) per assicurarti che esista una condizione di uscita realistica.
- Variabili di Controllo: Assicurati che le variabili utilizzate nei loop vengano modificate in modo tale da poter eventualmente soddisfare la condizione di uscita.
- Controllare che i cicli abbiano delle condizioni di uscita fattibili

6.Esempi vari

es: Programma moltiplicatore tra due numeri, tecnica moltiplicazione iterativa

```
ORG 100          //Il programma inizia a memoria 100
LDA IN2MUL       //Carica l’indirizzo di memoria IN2MUL
SZA              //Salta se il registro accumulatore zero
BUN NOZ          //Salta a “NOZ” se accumulatore è zero
HLT
```

```
NOZ, LDA IN1MUL //Carica IN1MUL nell'accumulatore se IN2MUL non è zero
SZA             //Salta se il registro accumulatore è zero
BUN OK          //Salto a "OK" se l'accumulatore
HLT
LDA IN2MUL      //Carica il contenuto di IN2MUL nell'accumulatore
CMA             //Complementa l'accumulatore
INC             //Incrementa l'accumulatore
ADD IN1MUL      //Aggiunge il contenuto di IN1MUL nell'accumulatore
SNA             //Salta se l'accumulatore non è zero
BSA SNAP   //Se l'accumulatore non è zero il programma salta a "SNAP" e salva AC
START, LDA IN1MUL   //Etichetta per il salto, salva IN1MUL nell'AC
CIR             //Shift logico a destra
STA IN1MUL      //Memorizza l'AC in IN1MUL
SZE             //Salta se zero
BSA UPD         //Salta ad UPD
CLE             //Cancella il registro E
LDA IN2MUL      //Carica nell'AC il contenuto di IN2MUL
CIL             //Shift logico a sinistra
STA IN2MUL      //Memorizza l'AC in IN2MUL
LDA IN1MUL      //Carica il contenuto di IN1MUL nell'AC
SZA             //Salta se l'AC è 0
BUN START       //Salta a "START" se l'AC non è zero
HLT
IN1MUL, DEC 3   //Definisce IN1MUL come il valore ottale 3
IN2MUL, DEC 10  //Definisce IN2MUL come il valore ottale 10
OUTMUL, DEC 0   //Definisce OUTMUL come il valore ottale 0
SWAP, DEC 0     //Definisce SWAP come il valore ottale 0
LDA IN1MUL      //Carica il contenuto di IN1MUL nell'AC
STA TMP         //Memorizza l'AC in TMP
LDA IN2MUL      //Carica il contenuto di IN2MUL nell'AC
STA IN1MUL      //Memorizza AC in IN1MUL
LDA TMP         //Carica il contenuto di TMP nell'AC
STA IN2MUL      //Memorizza l'AC in IN2MUL
BUN SWAP I      //Salta all'indirizzo indicato da "SWAP"
TMP, DEC 0      //Definisce TMP come il valore dell'ottale 0
UPD, DEC 0      //Definisce UPD come il valore dell'ottale 0
LDA IN2MUL      //Carica il contenuto di IN2MUL nell'AC
ADD OUTMUL      //Aggiunge OUTMUL all'AC
STA OUTMUL      //Memorizza l'AC in OUTMUL
BUN UPD I       //Salta ad UPD
END
```

Il programma gestisce i casi in cui uno dei due moltiplicandi è 0, restituisce 0, e il programma termina immediatamente.

L’approccio principale utilizza un ciclo di iterazioni controllate da istruzioni di salto condizionato e istruzioni di shift per eseguire le moltiplicazioni, durante ogni iterazione, uno dei moltiplicandi viene modificato mediante operazioni di shift a destra e a sinistra, mentre l’altro moltiplicando viene aggiunto ripetutamente al risultato accumulato nell’AC.
Il ciclo continua fino a quando il moltiplicando modificato diventa zero, a questo punto, il programma termina l’iterazione e il risultato finale del prodotto è contenuto nell’AC.

es: Scrivere il contenuto dell’AC dopo ogni istruzione eseguita nel seguente programma, nel caso in cui le celle di memoria etichettate x siano dec 106, hex 106

DEC 106:

```
ORG 100
CLA          //L’AC viene azzerato, AC=0
LDA X        //Carico in AC x, quindi AC=x
ADD Y        //Aggiungo y all’AC, quindi AC=x+y
CIL          //Sposto a sx di 1, il primo 1 dall’AC si toglie e va a finire nel registro E
BSA CHK  //Il contenuto dell’AC non cambia e salta all’indirizzo successivo
INC          //Sommo 1 all’AC
HLT          //Si ferma, il contenuto è lo stesso
CHK, HEX 0
SNA
BUN CHK I  //Salta incondizionatamente, ma il contenuto di AC è lo stesso
X, DEC 106
Y, HEX BF7F
END
```

HEX 106: //105 e 106 sono le zone di memoria

```
ORG 100
CLA              //L’AC viene azzerato, AC=0
LDA X            //Carico in AC x, quindi AC=x
ADD Y            //Aggiungo y all’AC, quindi AC=x+y
CIL              //Sposto a sx di 1, il primo 1 dall’AC si toglie e va a finire nel registro E
BSA CHK      //Il contenuto dell’AC non cambia e salta alla cella di memoria di x
105,INC          //Sommo 1 all’AC
106,HLT          //Si ferma, il contenuto è lo stesso
CHK, HEX 0   //
SNA
BUN CHK I  //Salta incondizionatamente, ma il contenuto di AC è lo stesso
```

X, HEX 106 //Salto qui, questi 3 bit li prende come istruzione, prendendo i 3 bit del codice operativo vede che sono tre 0, i quali corrispondono all'and bit a bit , quindi è minore di 7.
Fa l'and bit a bit tra il contenuto dell'AC e il contenuto della cella di memoria 106

Y, HEX BF7F //Stessa cosa di sopra, ma stavolta il codice operativo è 011, che comunque è <7, va a cercare nella parte della memoria F7F ma che non conosciamo, quindi va in crash, ma il contenuto dell'AC non cambia

END

<u>es Primo appello, conti giusti(?)</u>

```
ORG 209
209 LDA X I            //AC:0001001000010100=4628
20a STA X I
20b ADD X    <-puntato da X AC:4628->Aggiungo 523 ad ogni ciclo
20c SNA
20d BUN X I  <-All'ultimo ciclo quando esco AC=-32665
20e CMA      //AC:32665
20f CIL      //AC:-205
210 STA X
211 Y, ISZ X
212 BUN Y
213 HLT
214 X,DEC 523 //DOPO 210 X, DEC -205
END
```

```
523->base 16
523|b
 32|0
  2|2
```

523=20b

```
532->base 2   //532 è l'indirizzo di x in base 10
532 0
266 0
133 1
 66 0
 33 1
 16 0
  8 0
  4 0
```

2 0
1 1

1000010100 = 532
ADD X = 0001001000010100 = 4096+532 = 4628 //Se c'è l'ADD si aggiunge 4096
20b=0001001000010100 = 523

Numero MAX(2^15)-1=32767
Numero MIN(-2^15)=-32768

32767-4628=28140 //Quanto devo aumentare per arrivare all'overflow
28140/523=53 //Il ciclo 20c si ripete 53 volte prima di arrivare all'overflow
32870-32767=103 //Vedo quanto ho overflowato, sapendo che subito prima
//dell'overflow AC= 32347

-32768+103=-32665 //Calcolo l'AC dopo l'overflow
32665=0111111110011001 //Vedo quanto vale in binario dopo la CMA, visto che la
// CMA complementa me ne sbatto il topocazzo del neg
CIL:1111111100110010
0000000011001101 //Complemento a 2 per capire che numero è
1+4+8+64+128=205

Contenuto finale di AC=-205 //Dato che ISZ fa ++ fino a che non si ha 0, da
//-205 a 0 ci sono 205 passaggi
Il ciclo si ripete 205 volte
HEX 211=205

Istruzioni:
5+(3*20b)+5+(2*HEX 211) //20b=Volte che si ripete il ciclo in 20b,
//HEX 211=Volte che si ripete il ciclo in 211
//i 5 sono le istruzioni fuori dai cicli

10+(3*53)+(2*205)=10+159+410=579

-Quante operazioni svolge? 579
-Qual'è il contenuto dell'ac alla fine? -205

Codice istruzione:

I	OPR	ADDRESS
1 se indiretto sennò 0	3 bit di codice dell' struzione	Indirizzo cella a cui punta

CONVERSIONI

1.Come rappresentare un numero in binario

Diviso per 2 e leggo i resti dal basso verso l'alto
es: 263

263/2 = 131 resto 1 ^ 263 = 100000111
131/2 = 65 resto 1 |
65/2 = 32 resto 1 |
32/2 = 16 resto 0 |
16/2 = 8 resto 0 |
8/2 = 4 resto 0 |
4/2 = 2 resto 0 |
2/2 = 1 resto 0 |
1/2 = 0 resto 1 |

3.Come calcolare numero dec da binario

Uso le potenze del due partendo da dx, soltanto quando incontro un 1

es : 11100 2^2 + 2^3+ 2^4 = 28

4.Come rappresentare un numero <1 in binary representation

Moltiplico per due e segno la cifra prima della virgola, fino a quando non trovo uno schema ciclico, leggo i numeri segnati dall'alto verso il basso
es. 0.3

0.3x2 = 0.6 segno 0 0.3 = 010011|0011|0011…..
0.6x2 = 1.2 segno 1 Ho separato le parti che si ripetono all'infinito
0.2x2 = 0.4 segno 0
0.4x2 = 0.8 segno 0
0.8x2 = 1.6 segno 1
0.6x2 = 1.2 segno 1

Ho ritrovato 0.6 quindi da qui in poi ripeterà sempre come sopra

Importante. Cosa è un numero subnormal?

Numeri che in valore assoluto sono minori di 0,00006103515625

Iella 5.Convertire un numero NON SUBNORMAL in IEEE 754

es. 263.3
263 = 100000111
0.3 = 010011 0011 0011…

263.3 = 100000111.010011 0011 0011…..

1.0000011101 0011 0011… Ho shiftato il "." di 8 posizioni, fino alla prima cifra
1.|0000011101 0011 0011|… x 2^8 Moltiplico per 2^numero posizioni dello shift

Tutto ciò che sta dopo il "." è la mantissa, (per indicarla, l'ho messa tra "|")

La rappresentazione che sto cercando è del tipo:

X XXXXXXXX XXXXX…XX
Il primo bit è il bit del segno, se 0 il numero è positivo, altrimenti 1 se è negativo, in questo caso, visto che il numero che cerco è 263.3, il primo bit sarà 0

I successivi 5(per 16bit) oppure 8(per 32 bit) sono i bit dell'esponente, questi li trovo sommando il numero di shift cioè 8, a 15 (se a 16 bit) oppure a 127 (se a 32 bit) ottenendo 135, che converto in binario.

I restanti sono 23 fraction bits (per il 32 bit) , non sarebbero altro che la mantissa (10 per il 16 bit e non 23, tocca trovare conferme)

135 = 10000111

X XXXXXXXX XXXXXXXXXXXXXXXXXXXXXX

0 10000111 00000111010011001100110

Quindi questa è la rappresentazione di 263.3 in IEEE754

es: -263

263= 100000111
Per ottenere 1,00000111 faccio 8 shift
sommo 15 a 8 per ottenere l' esponente, quello dopo la virgola è la mantissa, il primo numero è 1 perchè il numero da rappresentare è negativo

1 10111 0000011100
(bit segno) (15+8= 23) (mantissa e aggiungo 2 0 alla fine per ottenere 10 cifre)

Quindi -263 in IEEE 754 è 1 10111 0000011100

6.Come convertire numero negativo con virgola NON SUBNORMAL in IEEE754

es: -0.001674112

-Numero negativo —–> bit segno = 1
-Iniziamo a moltiplicare il numero in valore assoluto per due fino a quando non troviamo un 1, una volta trovato l' 1 moltiplichiamo altre 10 volte

0.001674112 · 2 = 0.003348224

0.003348224 · 2 = 0.006696448

0.006696448 · 2 = 0.013392896

0.013392896 · 2 = 0.026785792

0.026785792 · 2 = 0.053571584

0.053571584 · 2 = 0.107143168

0.107143168 · 2 = 0.214286336

0.214286336 · 2 = 0.428572672

0.428572672 · 2 = 0.857145344

0.857145344 · 2 = 1.714290688

0.714290688 · 2 = 1.428581376

0.428581376 · 2 = 0.857162752

0.857162752 · 2 = 1.714325504

0.714325504 · 2 = 1.428651008

0.428651008 · 2 = 0.857302016

0.857302016 · 2 = 1.714604032

0.714604032 · 2 = 1.429208064

0.429208064 · 2 = 0.858416128

0.858416128 · 2 = 1.716832256

0.716832256 · 2 = 1.433664512

Riscrivo il numero decimale in forma binaria 0.00000000011011011011, successivamente lo scrivo come moltiplicazione di una potenza del 2:
$1{,}1011011011 \times 2^{-10}$

A quel punto l' esponente sarà 15 - l' esponente del 2(quante volte ho moltiplicato prima di trovare l'1 + 1) in questo caso (15-10) = 5
La mantissa è tutto ciò dopo la virgola, il risultato finale è
1 00101 1011011011

7.Come calcolare un numero SUBNORMAL in IEEE754

Se troviamo per esempio -0,00004, numero subnormal, l' esponente è sempre 00000, per trovare la mantissa bisogna applicare la regola della moltiplicazione per 2 per trovare il numero in binario e ripeterla 24 volte, a quel punto scarto i primi 14 numeri e prendo i restanti 10 come mantissa

es:

-0,00004

Bit segno: 1

Esponente: 00000

Mantissa:

0.00004·2=0.000080.

00008·2=0.00016

0.00016·2=0.00032

0.00032·2=0.00064

0.00064·2=0.00128

0.00128·2=0.00256

0.00256·2=0.00512

0.00512·2=0.01024

0.01024·2=0.02048

0.02048·2=0.04096

0.04096·2=0.08192

0.08192·2=0.16384

0.16384·2=0.32768

0.32768·2=0.65536

0.65536·2=1.31072

0.31072·2=0.62144

0.62144·2=1.24288

0.24288·2=0.48576

0.48576·2=0.97152

0.97152·2=1.94304

0.94304·2=1.88608

0.88608·2=1.77216

0.77216·2=1.54432

0.54432·2=1.08864

quindi il risultato è 1000001010011111

8.Come convertire un numero IEEE754 subnormal (ha tutti 0 all'esponente) in decimale

The exponent contains only zeros. Thus '0.' is prepended to the mantissa:

$0.1100001111 \cdot 2^{-14}$

Shift comma by 14 digits to the left:

0.00000000000000011000011111

Convert mantissa to decimal number:

$$0.00000000000000011000011111_2$$

$$\triangleq 2^{-15} + 2^{-16} + 2^{-21} + 2^{-22} + 2^{-23} + 2^{-24}$$

= 0,000030517578125 + 0,0000152587890625 + 0,000000476837158203125 + 0,000000238418579101562 5 + 0,000000119209289550781 25 + 0,000000059604644775390625

= 0.00004667043685913085937 5

9.Come convertire stringa binaria in numero formato IEEE 754

Bisogna seguire la seguente formula : (-1)^S x (1,M) x 2^(E-K)

https://www.h-schmidt.net/FloatConverter/IEEE754.html

S : Il primo bit della mia stringa binaria
M : Mantissa
E : Esponente
K : 15

es: 1011101101101010

1 | 01110 | 1101101010

S : 1
M : 1101101010
E : 01110 = 14

/-1* 1,1101101010 *2^(01110-15)
-1 * 0,11101101010
-1*(2^-1+2^-2+2^-3+2^-5+2^-6+2^-8+2^-10) = -0,936

Esempio: Rappresentare il numero -0.75 nel formato IEEE 754 a 16 bit

1. **Determinare il segno:**
   - -0.75 è negativo, quindi il bit di segno (S) è `1`.
2. **Convertire -0.75 in forma binaria normalizzata:**
   - 0.75 in binario è `0.11`.
   - In forma normalizzata: 1.1×2−1
3. **Calcolare l'esponente:**
   - L'esponente effettivo è `-1`.
   - L'esponente rappresentato EEE è −1+15=14-1 + 15 = 14−1+15=14.
   - 14 in binario è `01110`.
4. **Determinare la mantissa:**

- La mantissa è i bit dopo il punto decimale: `1000000000` (completato a 10 bit).

1011101000000000

10.Come calcolare numero in complemento a 2

Per capire il numero più grande che posso rappresentare con n bit seguo la seguente la formula:

2^(n-1) -1

<u>es: Trovare 12 in complemento a 2</u>

12 = 00001100

Come prima cosa inverto gli 0 e gli 1

11110011 a questo numero faccio +1

11110100 questo sarà -12

Quindi +12: 00001100 e invece -12: 11110100

11.Come calcolare un numero in modulo e segno

Considero il valore assoluto del numero con cui devo lavorare, lo rappresento in notazione binaria, e davanti metto il bit del segno

<u>es: Rappresenta -12 in modulo e segno</u>

12/2 = 6 resto 0 12 = 1100
6/2 = 3 resto 0
3/2 = 1 resto 1
1/2 = 0 resto 1

1 se è negativo e 0 se è positivo

-12 = 11100

12.Come rappresentare un numero da base 10 a base n

Divido il numero che voglio convertire per la base n che voglio

<u>es: Convertire 54 in base 4</u>

54/4 = 13 resto 2
13/4 = 3 resto 1
3/4 = 0 resto 3

54 = 312 in base 4

Al contrario

122 base 3 = 2x3^0 + 2x3^1 + 1x3^2 = 17 base 10

13.Come convertire un numero da decimale a esadecimale

es : 110110110 = 438
“:” = diviso

438 | 6
27 B
1 1
0

438 : 16 = 27,375
0.375 x 16 = 6
27 : 16 = 1,6875
0.6875 x 16 = 11 -> B
1 : 16 = 0.0625
0.0625 x 16 = 1

Ora leggo dal basso verso l’alto la colonna dei resti, quindi 1B6 è l’esadecimale di 438

14.Come trovare numero da stringa binaria con complemento a 2

1011101101101010

Inverto i bit
0100010010010101

Aggiungo 1
0100010010010110
Calcolo il valore di questa e poi ci metto il “-” davanti visto che la stringa iniziale era negativa

-17558

15.Trovare da stringa binaria a numero modulo e segno

1011101101101010

Considero il numero in valore assoluto, in questo caso levo dal cazzo il primo 1 che mi indica che il numero è negativo.
Quindi la mia stringa diventa:

011101101101010 = 15210
Di conseguenza, la stringa che cercavo varrà:

1011101101101010 = -15210

16.Trovare da stringa binaria a numero base 3

1011101101101010
Calcolo il valore della stringa binaria
1011101101101010

3^1+3^3+3^5+3^6+3^8+3^9+3^10+3^11+3^12+3^14 = 5577852

17.Convertire stringa binaria in assembly pdp8

Stringa binaria : 1011101101101010

Il primo 1 del bit mi indica che è indiretta

Successivamente prendo gli altri 3 bit che sono 011, codice operativo della STA (lo da lui), tutte le altre sono da convertire in esadecimale

101101101010 = B6A

Quindi mi diventa STA B6A I (nel caso il primo bit fosse stato 0, vuol dire che è diretta, in quel caso non metto la I alla fine)

CACHE

1.L'esercizio mi fornisce blocchi e posizioni

-Come prima cosa faccio 2^8 / n blocchi
-La parola / il numero che ho trovato sopra
-Quello che ho trovato sopra mod n posizioni

es:
Supponendo di avere una memoria centrale con indirizzi da 8 bit suddivisa in 128 blocchi e una memoria cache con indirizzamento DIRETTO con 4 posizioni, in che posizione di cache verrà memorizzata la parola di indirizzo 10010110?

n° parole = n° indirizzi / n° blocchi = 2^8 / 128 = 2 parole

10010110 = 2+4+16+128 = 150 indirizzo parola

150 / 2 = 75 indirizzo di blocco

75 mod 4 = 3

2.L'esercizio mi da blocchi, vie e gruppi

-Prima cosa trovo la parola
-Parola per blocco = 2^8 / n blocchi
-Blocco = Parola+1 / parole per blocco
-(Blocco-1) mod gruppi
-Quello sopra +1
es: Supponiamo di avere una memoria centrale con indirizzi da 8 bit suddivisi in 32 blocchi, e una memoria cache con indirizzamento associativo a 4 vie con 8 gruppi, dire in che posizione/i verrà creata la parola d'indirizzo 11000100

-Prima cosa da fare è capire il numero della parola

11000100 = 1x2^2 + 1x2^6 + 1x2^7 = 196 base 10
Parola = 196+1 = 197esima parola

-Calcoliamo quante parole per blocco sono presenti
2^8 (2^n.bit) = 256 / 32 = 8 parola per blocco // parola per blocco = (2^n bit) / n blocchi

-Divido il numero della parola per il numero delle parole per blocco
197/8 = 24,625 quindi 25esimo blocco, ossia blocco n 24 (approssimo sempre per eccesso)

24 mod 8 = 0 cioè il 1

Verrà creata nel 1 gruppo posizioni 0, 1 ,2, 3

es: Primo appello
Supponendo di avere una memoria centrale con 4096 indirizzi, suddivisa in blocchi di 8 parole, e una memoria cache associativa a 3 gruppi e 5 vie.
a)Quante parole di memoria può contenere la memoria cache?
b)A seguito di una evento di miss, quante parole in memoria cache verranno modificate?

c)In che posizione/i di cache può essere memorizzata la parola di indirizzo 195 base 16?

a)Capacità totale = 3 gruppi * 5 vie * 8 parole = 120 parole
b)Cache miss: In sostanza si va a perdere un intero blocco.
Essendo un blocco fatto da 8 parole, la risposta è : 8 parole
c)Posizioni: 10,11,12,13,14

195 base 16
16^2+9*16+5=405
405/8=50,625, quindi 51 esimo blocco, blocco numero 50
50 mod 3 = 2 quindi blocco 3
10, 11, 12, 13, 14

3.L'esercizio mi fornisce parole, vie e gruppi

es:
Supponendo di avere una memoria centrale con 256 indirizzo da 8 bit suddivisa in blocchi di 8 parole, e una memoria cache a 3 vie con 3 gruppi, in che posizione/i di cache verrà cercata la parola il cui indirizzo espresso in IEEE754 è 0101001100100000

0 10100 1100100000

\+ 2^20 - 2^15 1, 2^-1+2^-2+2^-5 = 2^5 x 1,78125 = 57 = 58 parola

58 / 8 = 7,25 = 8 blocco = blocco n 7

7 mod 3 = 1 = 2 gruppo

Posizioni 3,4,5

4.Esercizi con indirizzamento associativo

-Quando parliamo di cache indirizzamento associativo basta che scrivo tutte le posizioni da 0 a x-1

es: Supponendo di avere una memoria centrale con indirizzi a 8 bit, suddivisa in 64 blocchi, e una memoria cache con indirizzamento associativo con 8 posizioni, in che posizioni di cache verrà ricercata la parola di indirizzo 10111101

0 1 2 3 4 5 6 7