Lezione 20 Z64 6 + Z64 UC

giovedì 16 novembre 2023 12:11

Andiamo a vedere in dettaglio quali sono le istruzioni nello z64 (li suddividiamo in gruppi):

1. B
   a. Operando è un registro di uso generale , in indirizzo di memoria o un valore immediato.
2. E
   a. Operando è registro di uso generale o indirizzo di memoria.
3. G
   a. Operando di uso generale
4. K
   a. Operando è costante di 32 bit unsigned
5. M
   a. Operando è locazione di memoria , codificata come uno spiazzamento a partire da dal contenuto di RIP, dopo esecuzione fase fetch (punta a prossima istruzione):
   b. In dettaglio:

Rip punta a questo indirizzo	Dopo fetch punta qui					

   c. Ma cosa succede se il programma implementa un if ? (**si parla di salti**) : nel nostro caso si usa istruzione jump (salta ad un certo punto del programma : cambia indirizzo che non è sequenziale) . Con il jump la distanza in byte si calcola con lo spiazzamento e non con indirizzo completo.
6. K:
   a. Costante immediata a 32 bit .

Andiamo ora a vedere effettivamente le classi di operazioni dello z64 : ricordiamo che sono 8

1. Classe 0 : Controllo hardware
   a. Modificano lo stato della CPU , oppure eseguono istruzioni particolari
   b.

Tipo	Mnemonico	Operandi	O S Z P C	Descrizione
1	hlt	–	- - - - -	Mette la CPU in modalità di basso consumo energetico, finché non viene ricevuta l'interruzione successiva
2	nop	–	- - - - -	Nessuna operazione
3	int	Imm8	- - - - -	Chiama esplicitamente un gestore di interruzioni

   c. HTL= arresta : **Disattiva il ciclo decode-fetch-execute.**
   d. NOP = consuma byte: in exec non fa niente . Inserisce spazio in parti diverse del codice **. Serve per riallineare le istruzioni**
   e. INT= sincronizzazione per attivare gestore di dispositivi
   f. **Richiamo di programma : STACK**
      i. Struttura dati di tipo LIFO : Least in first out : inserisco e prelevo (ultimo) elemento .
      ii. In dettaglio:

El n-1
…
El 2
El 1
El 0

      iii. Con push inserisco elemento, con pop prelevo ultimo elemento e lo invalido.
      iv. Viene usata in quanto i registri sono limitati
      v. In generale è composto da quad-word (64 bit -> 8 byte) : se di dimensione minore, il processore lo estende
      vi. Per sapere quale è elemento affiorante della nostra pila : **Lo Stack Pointer** .

Mantiene al suo interno indirizzo elemento affiorante. Non viene usato esplicitamente dal programmatore . Questo stack di programma viene allocato alla fine della memoria (elemento più grande), quindi per inserire nuovo elemento , lo devo decrementare.

2. Classe 1 : istruzioni di movimento dati

a.

Tipo	Mnemonico	Operandi	O S Z P C	Descrizione
0	mov	B, E	- - - - -	Fa una copia di B in E
1	movsX	E, G	- - - - -	Fa una copia di E in G con estensione del segno
2	movzX	E, G	- - - - -	Fa una copia di E in G con estensione dello zero
3	lea	E, G	- - - - -	Valuta la modalità di indirizzamento, salva il risultato in G
4	push	E	- - - - -	Copia il contenuto di E sulla cima dello stack
5	pop	E	- - - - -	Copia il contenuto della cima dello stack in E
6	pushf	–	- - - - -	Copia sulla cima dello stack il registro FLAGS
7	popf	–	- - - - -	Copia nel registro FLAGS il contenuto della cima dello stack
8	movs	–	- - - - -	Esegue una copia memoria-memoria
9	stos	–	- - - - -	Imposta una regione di memoria ad un dato valore

b. Mov
   i. Operando da B ad E : copia soltanto
   ii. Attenzione al contesto : esistono varie istruzioni per fare la stessa operazione
   iii. MOVsX :
      1) Effettua la copia + estensione del segno (attenzione se aritmetica a CP2)
   iv. MOVzX:
      1) Estensione con lo "0" : interi postivi

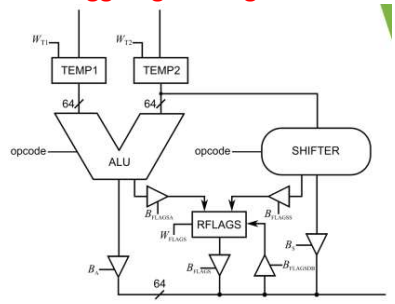
v.

Istruzione	Tipo di conversione
movsbw %al, %ax	Estendi il segno da byte a word
movsbl %al, %eax	Estendi il segno da byte a longword
movsbq %al, %rax	Estendi il segno da byte a quadword
movswl %ax, %eax	Estendi il segno da word a longword
movswq %ax, %rax	Estendi il segno da word a quadword
movslq %eax, %rax	Estendi il segno da longword a quadword

c. LEA (load effective address)
   i. Calcola indirizzo effettivo :
      1) Movq disp(%rax,%rcx,8),%rdx
         a) Cosi accedo a Base+(indice*scala)+offset
         b) Prende il valore contenuto
      2) Lea : valuta modalità indirizzamento e scrive nel registro il valore dell'indirizzo
         a) Lea disp(%rax,%rcx,8),%rcx
         b) Non accedo in memoria per sapere il contenuto in memoria .

d. Torniamo allo stack : PUSHF e POPF: permettono di effettuare una copia del registro flags (non modificabile dal programmatore) nello stack : devo quindi modificare hardware : aggiungo collegamento da data bus interno e flags .

i.



e. MOVS (move string) : istruzione di tipo Stringa : copia memoria-memoria (dimensione maggiore di 8 byte)
f. STOS (store string) : inizializza area di memoria ad un valore scelto dal programmatore

3. Classe 2 :istruzioni logico aritmetiche :

a.

Tipo	Mnemonico	Operandi	O S Z P C	Descrizione
0	add	B, E	⇕ ⇕ ⇕ ⇕ ⇕	Memorizza in E il risultato di E + B
1	sub	B, E	⇕ ⇕ ⇕ ⇕ ⇕	Memorizza in E il risultato di E - B
2	adc	B, E	⇕ ⇕ ⇕ ⇕ ⇕	Memorizza in D il risultato di E + B + CF
3	sbb	B, E	⇕ ⇕ ⇕ ⇕ ⇕	Memorizza in D il risultato di E - (B + neg(CF))
4	cmp	B, E	⇕ ⇕ ⇕ ⇕ ⇕	Confronta i valori di B ed E calcolando E - B, il risultato viene poi scartato
5	test	B, E	⇕ ⇕ ⇕ ⇕ ⇕	Calcola l'and logico bit a bit di B ed E, il risultato viene poi scartato
6	neg	E	⇕ ⇕ ⇕ ⇕ ⇕	Rimpiazza il valore di E con il suo complemento a 2

b. Possibile interagire anche con il carry flag : ADC
   i. Consente di superare il limite imposto dal numero di bit dei registri
c. SBB (subtract with barrow):
   i. Sottrazione a precisione arbitraria
   ii.
```
movq $operand_1_high, %rax
movq $operand_1_low, %rbx
movq $operand_2_high, %rcx
movq $operand_2_low, %rdx
addq %rbx, %rdx
adcq %rax, %rcx
```
d. CMP (compare) : effettua di confronto tra dati : non serve inserire un comparatore , in quanto i processori effettuano il confronto come una sottrazione:
   i. A=B??
      1) Il processore esegue B-A : se vale 0 sono uguali : uso lo zero-flags!!
   ii. A>B
      1) Il processore esegue B-A<0 : se si minore
   iii. **Non salva il risultato nella destinazione** .
e. Test
   i. A=0? -> a&a =0 . Si punta sul riuso del processore , usando stessa circuiteria per varie operazioni
f. Neg : Cp2 del valore : se mancasse questa istruzione, non si potrebbe fare la sottrazione.

g.

Tipo	Mnemonico	Operandi	O S Z P C	Descrizione
7	and	B, E	0 ⇕ ⇕ ⇕ 0	Memorizza in E il risultato dell'and bit a bit tra B ed E
8	or	B, E	0 ⇕ ⇕ ⇕ 0	Memorizza in E il risultato dell'or bit a bit tra B ed E
9	xor	B, E	0 ⇕ ⇕ ⇕ 0	Memorizza in E il risultato dello xor bit a bit tra B edE
10	not	E	0 ⇕ ⇕ ⇕ 0	Rimpiazza il valore di E con il suo complemento a uno
11	bt	K, E	- - - - ⇕	Imposta CF al valore del K-simo bit di E (bit testing)

4. Classe 3 : istruzioni di rotazione e shift

a.

Tipo	Mnemonico	Operandi	O S Z P C	Descrizione
0	sal	K, G	⇕ ⇕ ⇕ ⇕ ⇕	Moltiplica per 2, K volte
1	sal	G	⇕ ⇕ ⇕ ⇕ ⇕	Moltiplica per 2, RCX volte
0	shl	K, G	⇕ ⇕ ⇕ ⇕ ⇕	Moltiplica per 2, K volte
1	shl	G	⇕ ⇕ ⇕ ⇕ ⇕	Moltiplica per 2, RCX volte
2	sar	K, G	⇕ ⇕ ⇕ ⇕ ⇕	Dividi (con segno) per 2, K volte
3	sar	G	⇕ ⇕ ⇕ ⇕ ⇕	Dividi (con segno) per 2, RCX volte
4	shr	K, G	⇕ ⇕ ⇕ ⇕ ⇕	Dividi (senza segno) per 2, K volte
5	shr	G	⇕ ⇕ ⇕ ⇕ ⇕	Dividi (senza segno) per 2, RCX volte
6	rcl	K, G	⇕ - - - ⇕	Ruota a sinistra, K volte
7	rcl	G	⇕ - - - ⇕	Ruota a sinistra, RCX volte
8	rcr	K, G	⇕ - - - ⇕	Ruota a destra, K volte
9	rcr	G	⇕ - - - ⇕	Ruota a destra, RCX volte
10	rol	K, G	⇕ - - - ⇕	Ruota a sinistra, K volte
11	rol	G	⇕ - - - ⇕	Ruota a sinistra, RCX volte
12	ror	K, G	⇕ - - - ⇕	Ruota a destra, K volte
13	ror	G	⇕ - - - ⇕	Ruota a destra, RCX volte

b. Attenzione al tipo : identifica due istruzioni diverse :

i. Se a sx non c'è differenza (aritmetico = logico) :
   1) 0011 | 1110 -> in entrambi mi darà 0110 | 1100 : faccio entrare il bit 0 a sx
ii. Se a dx :
   1) Attenzione al segno (aritmetico != logico) :
      a) 1000 | 0011 -> 1100 | 0001
iii. Nel caso di shift , per stabilire la posizione di quanto shiftare ci sono due modi : o lo passo come secondo parametro oppure uso il registro contatore

5. Classe 4 : Manipolazione di bit di flags

a.

Tipo	Mnemonico	Operandi	O S Z P C	Descrizione
0	clc	–	- - - - 0	Resetta CF
1	clp†	–	- - - 0 -	Resetta PF
2	clz†	–	- - 0 - -	Resetta ZF
3	cls†	–	- 0 - - -	Resetta SF
4	cli	–	- - - - -	Resetta IF
5	cld	–	- - - - -	Resetta DF
6	clo†	–	0 - - - -	Resetta OF
7	stc	–	- - - - 1	Imposta CF
8	stp†	–	- - - 1 -	Imposta PF
9	stz†	–	- - 1 - -	Imposta ZF
10	sts†	–	- 1 - - -	Imposta SF
11	sti	–	- - - - -	Imposta IF
12	std	–	- - - - -	Imposta DF
13	sto†	–	1 - - - -	Imposta OF

†: non esiste un'istruzione corrispondente nell'assembly x86

   b. Cl = clear
   c. No move possibile , ma cosi si

6. Classe 5 : istruzioni di controllo del flusso del programma

a.

Tipo	Mnemonico	Operandi	O S Z P C	Descrizione
0	jmp	M	- - - - -	Esegue un salto relativo
1	jmp	*G	- - - - -	Esegui un salto assoluto
2	call	M	- - - - -	Esegue una chiamata a subroutine relativa
3	call	*G	- - - - -	Esegue una chiamata a subroutine assoluta
4	ret	–	- - - - -	Ritorna da una subroutine
5	iret	–	⇕ ⇕ ⇕ ⇕ ⇕	Ritorna dal gestore di una interruzione

   b. Di solito esecuzione istruzione è sequenziale , ma possiamo alterare questo flusso : JMP e CALL . Nel primo si salta ad un indirizzo distante : il processore salta a punto esplicito (M=offset) : quindi il processore aggiorna RIP , non tenendo conto di indirizzo partenza ; nel secondo caso chiamo le sub-routine (il flusso di controllo viene trasferito a prima istruzione della sub-routine) . Il ritorno invece si ha ad istruzione dopo invocazione sub-routine (effettuata con istruzione ret); Iret variante che conclude interazione / esecuzione driver. Gli argomenti di jump e call possono essere indirizzo e/o registro . **Spiazzamento di RIP (salto relativo) / indirizzo a 64 bit nel registro (salto assoluto)** .

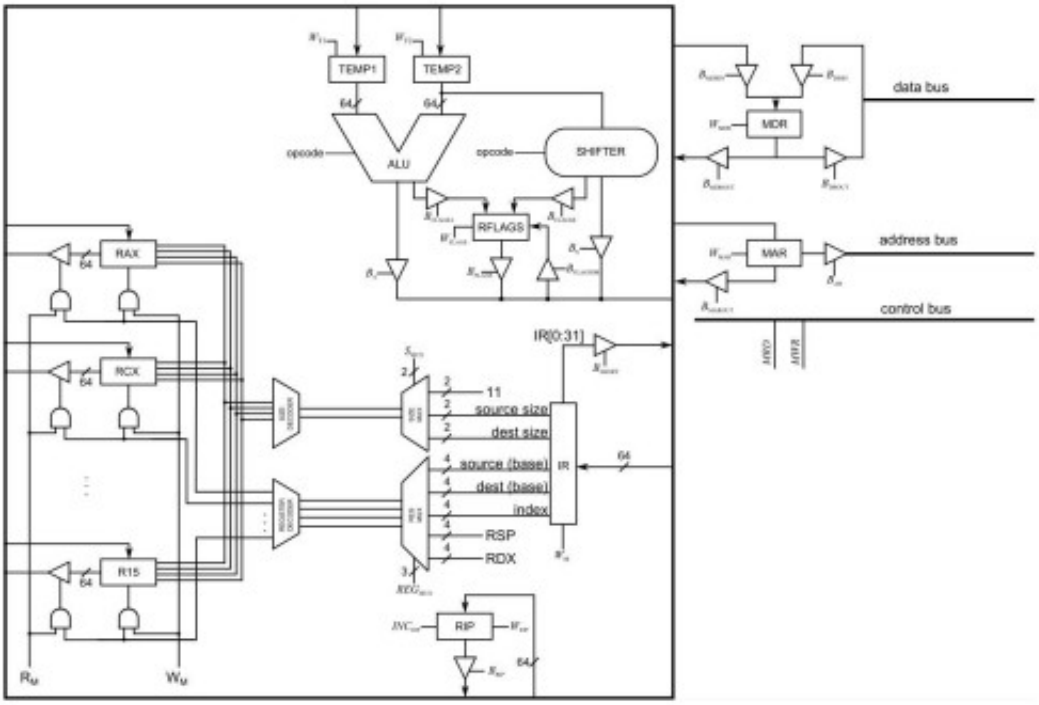
7. Classe 6 : Controllo condizionale del flusso

a.

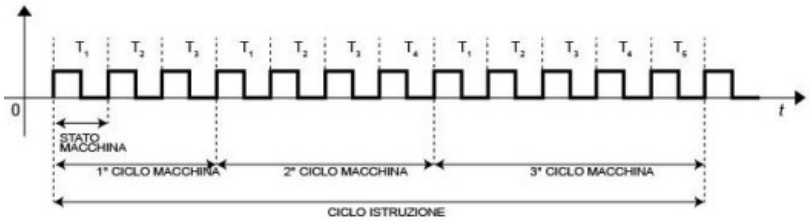
Tipo	Mnemonico	Operandi	O S Z P C	Descrizione
0	jc	M	- - - - -	Salta a M se CF è impostato
1	jp	M	- - - - -	Salta a M se PF è impostato
2	jz	M	- - - - -	Salta a M se ZF è impostato
3	js	M	- - - - -	Salta a M se SF è impostato
4	jo	M	- - - - -	Salta a M se OF è impostato
5	jnc	M	- - - - -	Salta a M se CF non è impostato
6	jnp	M	- - - - -	Salta a M se PF non è impostato
7	jnz	M	- - - - -	Salta a M se ZF non è impostato
8	jns	M	- - - - -	Salta a M se SF non è impostato
9	jno	M	- - - - -	Salta a M se OF non è impostato

   b. Se JNZ faccio salti se condizioni sono 0.

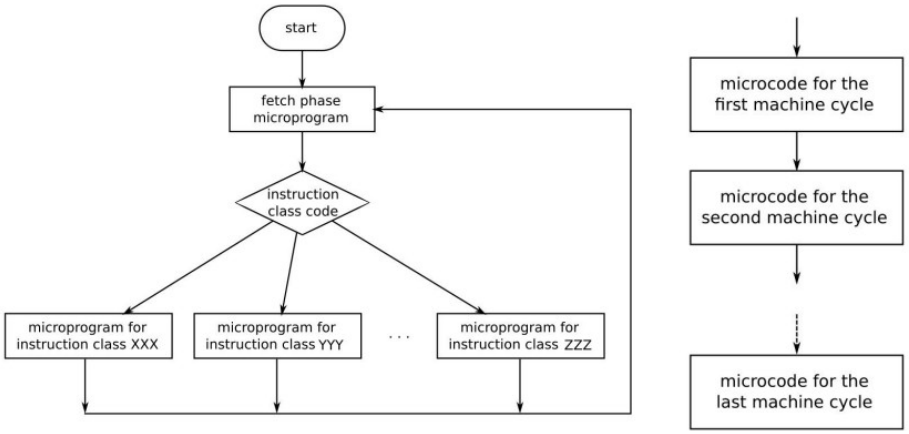
Quindi l'architettura finale dello z-64 è la seguente :



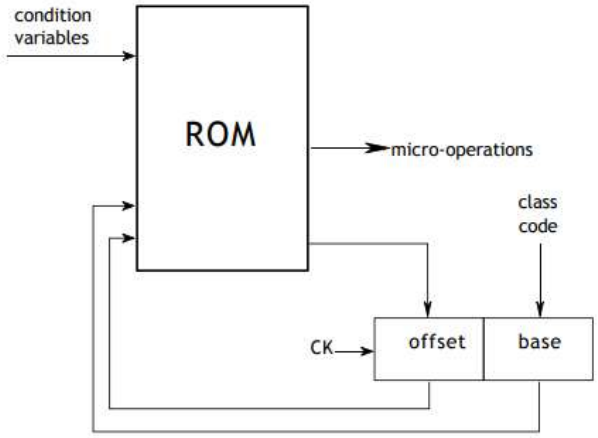
Andiamo ora a vedere l' unità di controllo : l'unità di controllo implementa le micro-operazioni : quindi unità di controllo è un'insieme di micro-programmi :



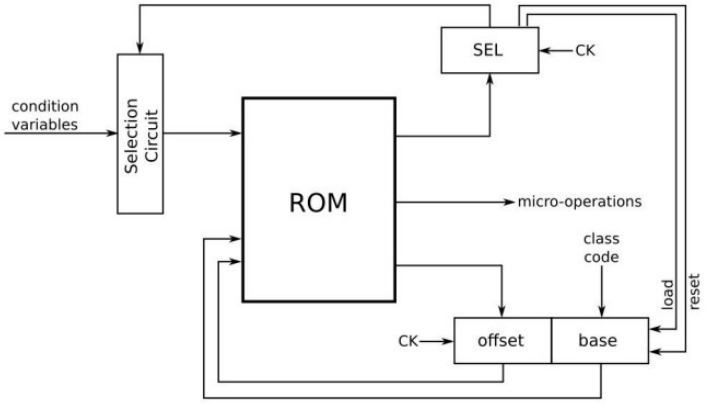
Quindi la CU è una grande macchina a stati finiti , la quale esegue micro-operazioni in più cicli macchina (visto che è multi ciclo) .In dettaglio :



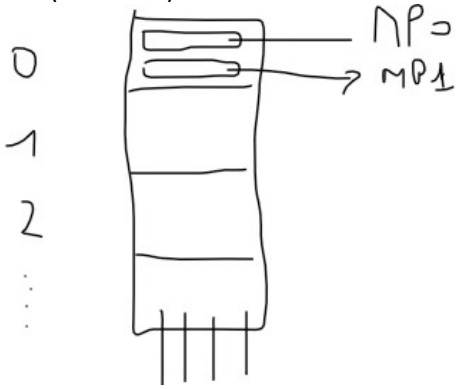
In dettaglio : gli input alla mia unità di controllo ha diverti input : IR , registro Flags (operazione si / no) , eventuali segnali da dispositivi esterni (uso data bus) : quindi gli input dipende da come abbiamo implementato l'automa , mentre per quanto riguarda gli output dipende da come è stata implementata l'unità di processamento . In dettaglio :



Ma questo schema ha un problema : quanti segnali in input ha ?? Andiamo a vederlo :



In questo circuito per circuito di selezione si intendono un numero di mux , il cui output è il massimo numero di variabili che vengono processate. Inoltre la rom fornisce i segnali di controllo per il selection. L'offset invece viene implementato come incremento, vista la sequenzialità delle micro-istruzioni. Non sempre rispetta questa meccanica, infatti se si parla di salti (esempio jz) , l'offset non è sequenziale .
Ricordiamoci della Rom Paginata(già vista) : la forma della Rom da rettangolare a quadrata: in ogni pagina ci vanno le micro istruzioni e per ogni riga di ogni pagina, ci sono le codifiche (circuitali) dei segnali di uscita (controllo) :



Dove a sx si intendono il numero delle pagine della Rom , mentre a destra si intendono le micro-istruzione del micro-programma contenuto nella pagine . Infine sotto ci sono i segnali di uscita , derivanti dalla realizzazione circuitale della ROM. Attenzione alla frammentazione interna : micro-istruzioni occupano dimensione minore di quella allocata alla pagina della ROM.