CESAR

(afinal as instruções)

Conjunto de Instruções

- 10 grupos de instruções
  - identificação pelos 4 bits mais significativos
  - tamanho
    - um byte
    - 2 bytes
- semelhante ao PDP-11

K. Thompson e D. Ritchie com um PDP-11

Identificação da instrução

4 bits mais significativos

Código	*Tipo*
0000	instrução de NOP
0001 e 0010	instruções sobre os códigos de condição
0011	instruções de desvio condicional
0100	instrução de desvio incondicional (JMP)
0101	instrução de controle de laço (SOB)
0110	instrução de desvio para subrotina (JSR)
0111	instrução de retorno de subrotina (RTS)
1000	instruções de um operando
1001 a 1110	instruções de dois operandos
1111	instrução de parada (HLT)

Instruções de 1 byte

Código	*Tipo*
0000	instrução de NOP
0001 e 0010	instruções sobre os códigos de condição
0011	instruções de desvio condicional
0100	instrução de desvio incondicional (JMP)
0101	instrução de controle de laço (SOB)
0110	instrução de desvio para subrotina (JSR)
0111	instrução de retorno de subrotina (RTS)
1000	instruções de um operando
1001 a 1110	instruções de dois operandos
1111	instrução de parada (HLT)

Instruções de 1 byte

	7							0
NOP	0	0	0	0	x	x	x	x

	7							0
CCC	0	0	0	1	n	z	v	c
SCC	0	0	1	0	n	z	v	c

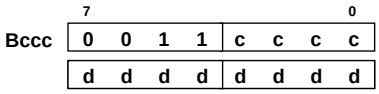
clear condition codes
set condition codes

	7							0
HLT	1	1	1	1	x	x	x	x

Desvios

Código	*Tipo*
0000	instrução de NOP
0001 e 0010	instruções sobre os códigos de condição
0011	instruções de desvio condicional
0100	instrução de desvio incondicional (JMP)
0101	instrução de controle de laço (SOB)
0110	instrução de desvio para subrotina (JSR)
0111	instrução de retorno de subrotina (RTS)
1000	instruções de um operando
1001 a 1110	instruções de dois operandos
1111	instrução de parada (HLT)

Desvio condicional



if cccc
then PC ← PC + ddddddddd

cccc = condição
ddddddddd = deslocamento

deslocamento : 8 bits em complemento de 2

Bccc foi pensado para ser usado após uma subtração

7

Condições para desvio

cccc	*mnemônico*	*Condição de desvio*
0000	BR (always)	sempre verdadeira
0001	BNE (Not Equal)	z = 0
0010	BEQ (EQual)	z = 1
0011	BPL (PLus)	n = 0
0100	BMI (MInus)	n = 1
0101	BVC (oVerflow Clear)	v = 0
0110	BVS (oVerflow Set)	v = 1
0111	BCC (Carry Clear)	c = 0
1000	BCS (Carry Set)	c = 1
1001	BGE (Greater or Equal)	n = v
1010	BLT (Less Than)	n < > v
1011	BGT (GreaTer)	n = v and z = 0
1100	BLE (Less or Equal)	n < > v or z = 1
1101	BHI (HIgher)	c = 0 and z = 0
1110	BLS (Lower or Same)	c = 1 or z = 1

8

Condições para desvio

cccc	*mnemônico*	*Condição de desvio*
0000	BR (always)	sempre verdadeira
0001	BNE (Not Equal)	z = 0
0010	BEQ (EQual)	z = 1
0011	BPL (PLus)	n = 0
0100	BMI (MInus)	n = 1
0101	BVC (oVerflow Clear)	v = 0
0110	BVS (oVerflow Set)	v = 1
0111	BCC (Carry Clear)	c = 0
1000	BCS (Carry Set)	c = 1

lembrar que:
uma subtração foi realizada anteriormente (a – b)
códigos de condição são verificados (n z v c)

9

Condições para desvio

cccc	*mnemônico*	*Condição de desvio*
1001	BGE (Greater or Equal)	n = v
1010	BLT (Less Than)	n < > v
1011	BGT (GreaTer)	n = v and z = 0
1100	BLE (Less or Equal)	n < > v or z = 1

vale para números em representação em complemento de 2

10

BGE & BLT

6 - 2	2 – (-6)	-2 – (-6)	2 - 6	-6 – (-2)
0110	0010	1110	0010	1010
0010	1010	1010	0110	1110
0110	0010	1110	0010	1010
+1110	+0110	+0110	+1010	+0010
0100	1000	0100	1100	1100
c = 1	c = 0	c = 1	c = 0	c = 0
n = 0	n = 1	n = 0	n = 1	n = 1
v = 0	v = 1	v = 0	v = 0	v = 0

exemplos com 4 bits

11

BGE & BLT

- operação anterior: (a – b)
- cc disponíveis: n z v c (após operação)
- c não serve (complemento de dois)
  - n=0 e v=0 → a ≥ b
  - n=0 e v=1 → a < b
  - n=1 e v=0 → a < b
  - n=1 e v=1 → a ≥ b

12

Condições para desvio

cccc	*mnemônico*	*Condição de desvio*
1101	BHI (HIgher)	c = 0 and z = 0
1110	BLS (Lower or Same)	c = 1 or z = 1

diferença entre *Higher* e *Greater*

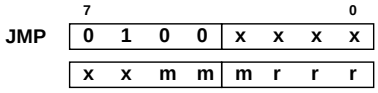
higher para números sem sinal

e o *Higher* or *Same ?*

Bcc

13

Desvio incondicional



PC ← endereço de desvio (modo 0 = NOP)

pode ser usado com qualquer modo de endereçamento exceto modo registrador (modo 0)

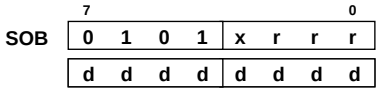
14

Controle de laço

Código	*Tipo*
0000	instrução de NOP
0001 e 0010	instruções sobre os códigos de condição
0011	instruções de desvio condicional
0100	instrução de desvio incondicional (JMP)
0101	instrução de controle de laço (SOB)
0110	instrução de desvio para subrotina (JSR)
0111	instrução de retorno de subrotina (RTS)
1000	instruções de um operando
1001 a 1110	instruções de dois operandos
1111	instrução de parada (HLT)

15

Controle de laço



r ← r - 1; if r<>0 then PC ← PC - d

subtrai um do registrador e desvia se não zero

idéia: voltar para início do laço

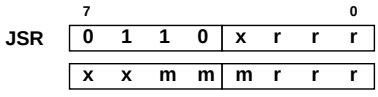
16

Subrotinas

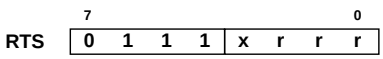
Código	*Tipo*
0000	instrução de NOP
0001 e 0010	instruções sobre os códigos de condição
0011	instruções de desvio condicional
0100	instrução de desvio incondicional (JMP)
0101	instrução de controle de laço (SOB)
0110	instrução de desvio para subrotina (JSR)
0111	instrução de retorno de subrotina (RTS)
1000	instruções de um operando
1001 a 1110	instruções de dois operandos
1111	instrução de parada (HLT)

17

Subrotina: desvio e retorno



temp ← endereço de desvio (modo 0 = NOP)
pilha ← registrador r
registrador r ← PC
PC ← temp



PC ← registrador r
registrador r ← pilha

mais detalhes serão discutidos posteriormente

18

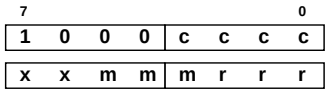
Instruções aritméticas

Código	*Tipo*
0000	instrução de NOP
0001 e 0010	instruções sobre os códigos de condição
0011	instruções de desvio condicional
0100	instrução de desvio incondicional (JMP)
0101	instrução de controle de laço (SOB)
0110	instrução de desvio para subrotina (JSR)
0111	instrução de retorno de subrotina (RTS)
1000	instruções de um operando
1001 a 1110	instruções de dois operandos
1111	instrução de parada (HLT)

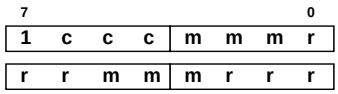
19

Aritméticas

um operando



dois operandos



primeiro operando = origem (ou fonte)
segundo operando = destino

20

Aritméticas com um operando

Obs: após uma subtração, C = 1 indica BORROW !

cccc	*instrução*	*significado*	*N*	*Z*	*C*	*V*
0000	CLR	op ← 0	t	t	0	0
0001	NOT	op ← NOT op	t	t	1	0
0010	INC	op ← op + 1	t	t	t	t
0011	DEC	op ← op - 1	t	t	not(t)	t
0100	NEG	op ← - op	t	t	not(t)	t
0101	TST	op ← op	t	t	0	0
0110	ROR	op ← SHR(c & op)	t	t	lsb	xor
0111	ROL	op ← SHL(op & c)	t	t	msb	xor
1000	ASR	op ← SHR(msb & op)	t	t	lsb	xor
1001	ASL	op ← SHL(op & 0)	t	t	msb	xor
1010	ADC	op ← op + c	t	t	t	t
1011	SBC	op ← op - c	t	t	t	t

21

Aritméticas com um operando

cccc	*instrução*	*significado*	*N*	*Z*	*C*	*V*
0000	CLR	op ← 0	t	t	0	0
0001	NOT	op ← NOT op	t	t	1	0
0010	INC	op ← op + 1	t	t	t	t

t – código de condição correspondente é testado e alterado pela unidade de controle

22

Aritméticas com um operando

cccc	*instrução*	*significado*	*N*	*Z*	*C*	*V*
0000	CLR	op ← 0	t	t	0	0
0001	NOT	op ← NOT op	t	t	1	0
0010	INC	op ← op + 1	t	t	t	t
0011	DEC	op ← op - 1	t	t	not(t)	t
0100	NEG	op ← - op	t	t	not(t)	t

Obs: após uma subtração, C = 1 indica BORROW !

not(t) – a ULA efetua operação de subtração pela soma do complemento de dois do subtraendo; neste caso o borrow e o inverso do carry

23

Aritméticas com um operando

cccc	*instrução*	*significado*	*N*	*Z*	*C*	*V*
0000	CLR	op ← 0	t	t	0	0
0001	NOT	op ← NOT op	t	t	1	0
0010	INC	op ← op + 1	t	t	t	t
0011	DEC	op ← op - 1	t	t	not(t)	t
0100	NEG	op ← - op	t	t	not(t)	t
0101	TST	op ← op	t	t	0	0

TST – testa N e Z sem armazenar o resultado
(o ciclo de escrita não é realizado, o operando não se altera, apenas os códigos de condição N e Z)

24

Aritméticas com um operando

xor – ou exclusivo entre os cc N e C

cccc	*instrução*	*significado*	*N*	*Z*	*C*	*V*
0000	CLR	op ← 0	t	t	0	0
0001	NOT	op ← NOT op	t	t	1	0
0010	INC	op ← op + 1	t	t	t	t
0011	DEC	op ← op - 1	t	t	not(t)	t
0100	NEG	op ← - op	t	t	not(t)	t
0101	TST	op ← op	t	t	0	0
0110	ROR	op ← SHR(c & op)	t	t	lsb	xor
0111	ROL	op ← SHL(op & c)	t	t	msb	xor
1000	ASR	op ← SHR(msb & op)	t	t	lsb	xor
1001	ASL	op ← SHL(op & 0)	t	t	msb	xor

lsb – bit menos significativo

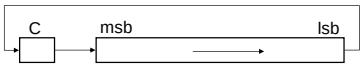
msb – bit mais significativo

25

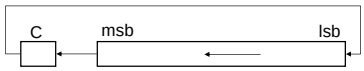
Rotação

& = concatenação

ROR: op ← SHR(c & op)



ROL: op ← SHL(op & c)

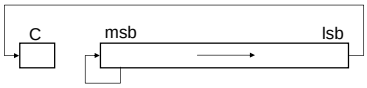


26

Deslocamento

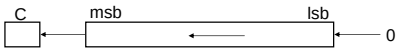
ASR: op ← SHR(msb & op)

equivale a dividir por 2 em complemento de 2



ASL: op ← SHL(op & 0)

equivale a multiplicar por 2 em complemento de 2



xor – ou exclusivo entre os cc N e C só faz sentido neste caso

27

Aritméticas com um operando

cccc	*instrução*	*significado*	*N*	*Z*	*C*	*V*
0000	CLR	op ← 0	t	t	0	0
0001	NOT	op ← NOT op	t	t	1	0
0010	INC	op ← op + 1	t	t	t	t
0011	DEC	op ← op - 1	t	t	not(t)	t
0100	NEG	op ← - op	t	t	not(t)	t
0101	TST	op ← op	t	t	0	0
0110	ROR	op ← SHR(c & op)	t	t	lsb	xor
0111	ROL	op ← SHL(op & c)	t	t	msb	xor
1000	ASR	op ← SHR(msb & op)	t	t	lsb	xor
1001	ASL	op ← SHL(op & 0)	t	t	msb	xor
1010	ADC	op ← op + c	t	t	t	t
1011	SBC	op ← op - c	t	t	t	t

28

Aritméticas com 2 operandos

ccc	*instrução*	*significado*	*N*	*Z*	*V*	*C*
001	MOV	dst ← src	t	t	0	-
010	ADD	dst ← dst + src	t	t	t	t
011	SUB	dst ← dst - src	t	t	t	not(t)
100	CMP	src - dst	t	t	t	not(t)
101	AND	dst ← dst AND src	t	t	0	-
110	OR	dst ← dst OR src	t	t	0	-

Primeiro operando = origem
Segundo operando = destino

Obs: após uma subtração, C = 1 indica BORROW !

29

SUB e CMP

ccc	*instrução*	*significado*	*N*	*Z*	*V*	*C*
001	MOV	dst ← src	t	t	0	-
010	ADD	dst ← dst + src	t	t	t	t
011	SUB	dst ← dst - src	t	t	t	not(t)
100	CMP	src - dst	t	t	t	not(t)
101	AND	dst ← dst AND src	t	t	0	-
110	OR	dst ← dst OR src	t	t	0	-

Primeiro operando = origem
Segundo operando = destino

SUB e CMP operam com ordem diferente

30

Exemplos de somas - codificar

ADD R2,R1	R1 = R1 +R2
ADD 2000,R1	R1 = R1 + MEM(2000)
ADD R2,3000	MEM(3000) = MEM(3000) + R2
ADD 2000,3000	MEM(3000) = MEM(3000) + MEM(2000)
ADD #5,R1	R1 = R1 + 5
ADD #300,1300	MEM(1300) = MEM(1300) + 300

31

Exemplos de somas - codificar

- R1 = R1 + Topo da pilha, sem retirar da pilha

  ADD (R6),R1
- R1 = R1 + Topo da pilha, retirando da pilha

  ADD (R6)+,R1
- Somar as duas palavras do topo da pilha, e devolver o resultado para o topo da pilha

  ADD (R6)+,(R6)

32