

Instruction Set Architecture Z01.1

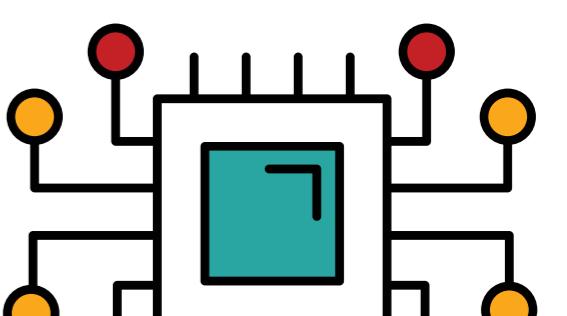
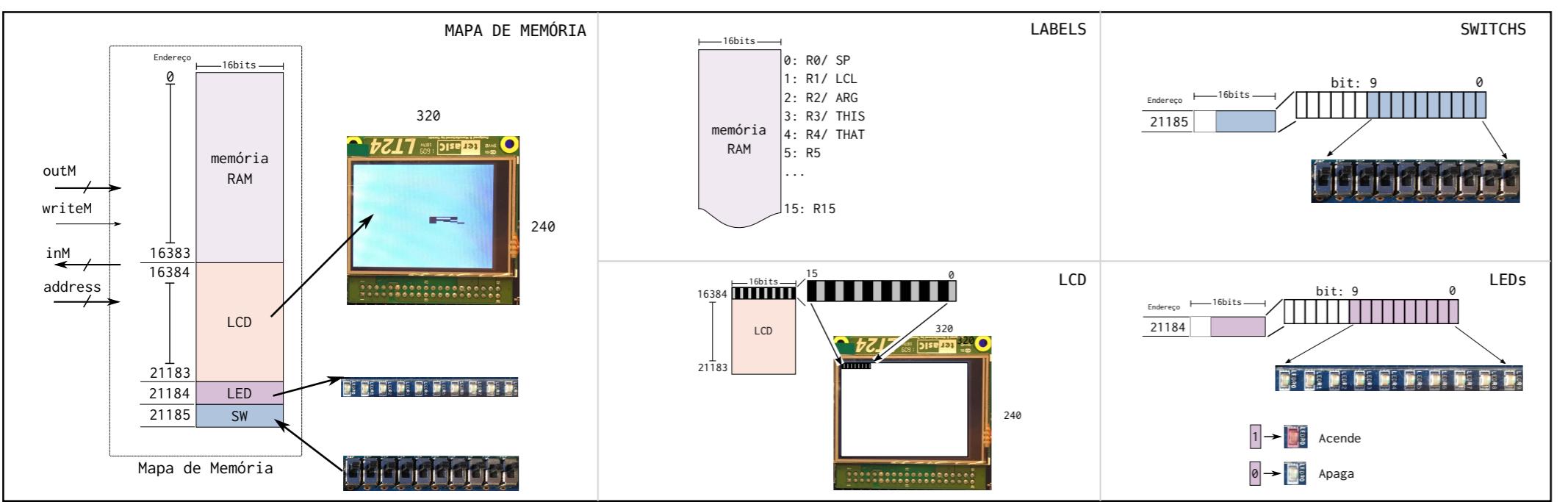
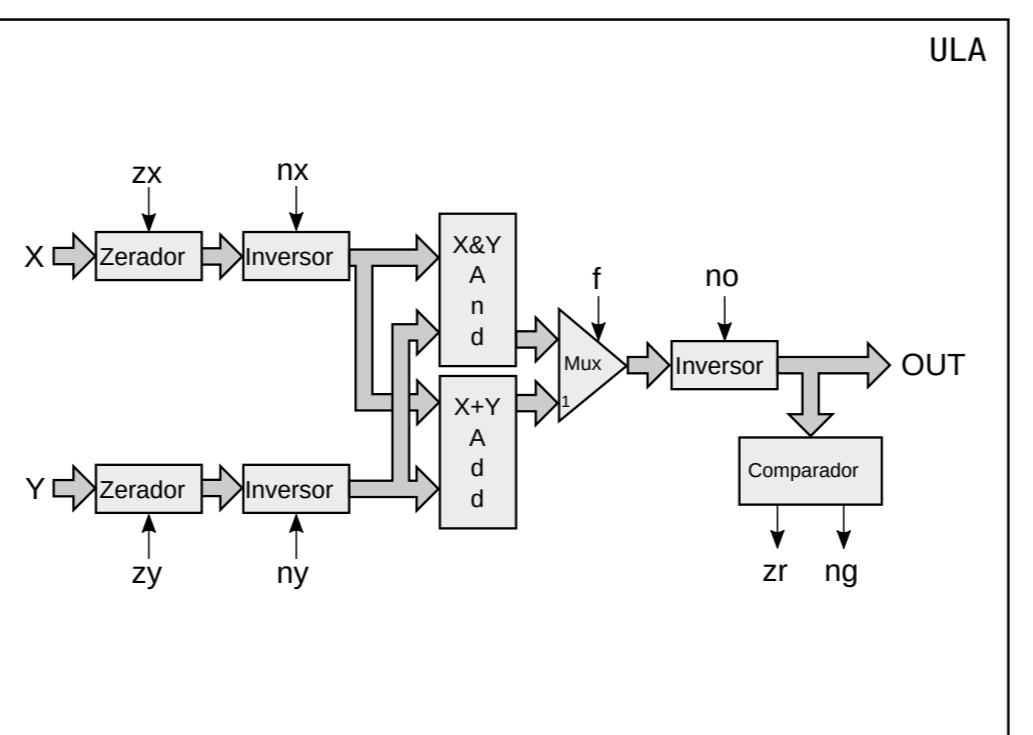
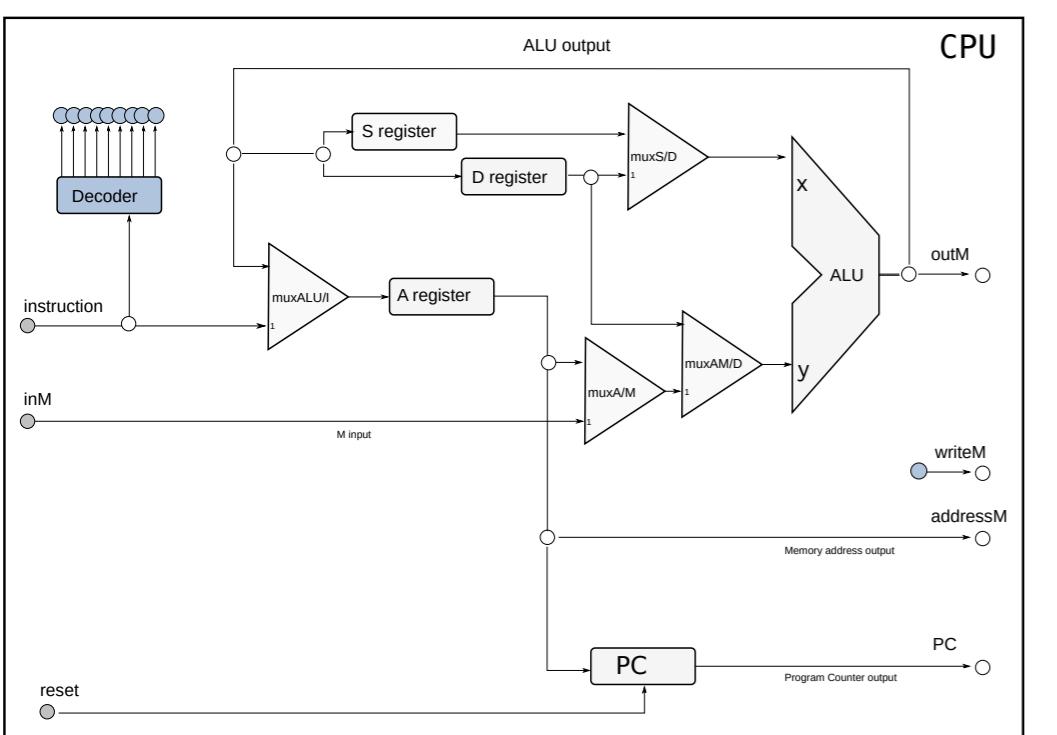
Versão Insper 2018b da arquitetura do livro : Elements of Computer System
Alteração:
Z01.1 : Possibilita operações entre S e D. Tamanho da instrução passou de 16 para 18 bits.
Z01 : Adicionou um registrador A mais na arquitetura chamado de registrador S. Para endereçar esse novo registrador, um novo bit d3 foi adicionado as instruções do tipo C.

Instruções do tipo A																	
se bit 17 == 0:																	
transfere 15 bits para o registrador A																	
17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	v	v	v	v	v	v	v	v	v	v	v	v	v	v	v
[15:0] : Palavra de 16 bits (positiva)																	

Instruções do tipo C																	
se bit 17 == 1:																	
executa ação																	
17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	X	r2	r1	r0	c5	c4	c3	c2	c1	c0	d3	d2	d1	d0	j2	j1	j0
[17:0] : Indica ação a ser executada pela CPU																	

Cálculo																			
$r2 = 0$	$r2 = 1$	zx	nx	zy	ny	f	no	zr	ng	$c5$	$c4$	$c3$	$c2$	$c1$	$c0$				
$r1 = 0$	$r1 = 0$	$r1 = 1$	$r1 = 1$	$r1 = 0$	$r1 = 0$	-	-	-	-	-	-	1	0	1	0	1	0	0	
$r0 = 0$	$r0 = 1$	$r0 = 0$	$r0 = 1$	$r0 = 0$	$r0 = 1$	-	-	-	-	-	-	1	1	1	1	1	1	1	
0	-	-	-	-	-	-	-	-	-	-	-	1	0	1	0	1	0	0	
1	-	-	-	-	-	-	-	-	-	-	-	1	1	1	1	1	1	1	
-1	-	-	-	-	-	-	-	-	-	-	-	1	1	0	1	0	0	0	
D	S	-	-	-	-	-	-	-	-	-	-	0	0	1	1	0	0	0	
A	-	(A)	-	-	-	-	-	-	-	-	-	1	1	0	0	0	0	0	
!D	!S	-	-	-	-	-	-	-	-	-	-	0	0	1	1	0	1	1	
!A	-	!(A)	-	-	-	-	-	-	-	-	-	1	1	0	0	0	1	1	
-D	-S	-	-	-	-	-	-	-	-	-	-	0	0	1	1	1	1	1	
-A	-	-(A)	-	-	-	-	-	-	-	-	-	1	1	0	0	1	1	1	
D+1	S+1	-	-	-	-	-	-	-	-	-	-	0	1	1	1	1	1	1	
A+1	-	(A)+1	-	-	-	-	-	-	-	-	-	1	1	0	1	1	1	1	
D-1	S-1	-	-	-	-	-	-	-	-	-	-	0	0	1	1	1	0	0	
A-1	-	(A)-1	-	-	-	-	-	-	-	-	-	1	1	0	0	1	0	0	
D+A	S+A	D+(A)	S+(A)	S+(A)	S+(A)	S+D	0	0	0	0	1	0	0	0	0	1	1	1	
D-A	S-A	D-(A)	S-(A)	S-(A)	S-(A)	S-D	0	1	0	0	1	1	1	0	0	1	0	1	
A-D	A-S	(A)-D	(A)-S	(A)-S	(A)-S	D-S	0	0	0	1	1	1	1	0	0	0	1	1	
D&A	S&A	D&(A)	S&(A)	S&(A)	S&(A)	S&D	0	0	0	0	0	0	0	0	0	0	0	0	
D A	S A	D (A)	S (A)	S (A)	S (A)	S D	0	1	0	1	0	1	0	1	0	1	1	1	

Destino									Jump								
$d3 = 0$	$d2 = 1$	$d1 = 0$	$d0 = 0$	$d3 = 1$	$d2 = 1$	$d1 = 0$	$d0 = 0$	$d3 = 0$	$d2 = 1$	$d1 = 0$	$d0 = 0$	$<0 = 0$	$j2$	$j1$	$j0$		
NULL	0	0	0	0	0	0	0	nada	0	0	0	0	0	0	0		
A	1	0	0	0	0	0	0	JG	0	0	0	1	0	0	0		
D	0	0	1	0	0	0	0	JE	0	0	1	0	0	0	0		
S	0	1	0	0	0	0	0	JGE	0	1	1	1	0	0	0		
(A)	0	0	0	1	0	0	0	JL	1	0	0	0	0	0	0		
D(A)	0	0	1	1	1	0	0	JNE	1	1	0	1	0	0	0		
SD	0	1	1	1	1	0	0	JLE	1	1	0	1	0	0	0		
A(A)	1	0	0	1	0	1	0	JMP	1	1	1	1	1	0	0		
AD	1	0	1	0	1	1	1										
AD(A)	1	0	1	1	1	1	1										
AS	1	1	0	0	0	0	0										
AS(A)	1	1	1	0	1	0	0										
ASD	1	1	1	1	0	0	0										
ASD(A)	1	1	1	1	1	1	1										



Elementos De Sistemas
Eng. Comp.
Prof. Rafael Corsi

Insper

Notações:

im : valor imediato (somente os valores 1, 0 e -1).

reg : registrador.

mem: memória, ou seja (%A).

Limitações:

- A arquitetura não permite somar o valor da memória apontada por (%A) com o valor de %A, ou de (%A) com (%A), tampouco %A com %A.
- Não é possível somar (ou subtrair, se é que isso faz sentido) o registrador com o mesmo, por exemplo somar %D com %D.
- Não é possível ler e gravar a memória ao mesmo tempo. Por exemplo, as instruções abaixo não funcionam no nosso computador:
 - incw (%A);
 - subw (%A),%D,(%A).

Observação:

A linguagem Assembly apresentada é específica para o processador produzido no curso. Embora muito similar a outras usadas em produtos de mercado, as instruções possuem limitações inerentes a cada hardware.

LEA - Carregamento Efetivo do Endereço (Valor)

`leaw im[15], reg[16]`

Descrição: A instrução `lea` armazena o valor passado (im[15]) no registrador especificado (na nossa implementação, somente o %A).

Exemplo : $A = 15$

`leaw $15,%A`

MOV - Cópia Valores

`movw im*/reg/mem, reg/mem {, reg/mem, reg/mem}`

(o imediato pode utilizar somente os valores: 1, 0 ou -1)

Descrição: A instrução `mov`, copia o valor da primeira posição para a segunda posição (terceira e quarta opcional).

Exemplo : $S = RAM/A$

`movw (%A),%S`

DEC - Decrementa Inteiro

`decw reg/mem`

Descrição: A instrução `dec`, subtrai um (1) do valor do registrador ou memória.

Exemplo : $A = A - 1$

`decw %A`

NOT – Negação por Complemento de Um

`notw reg/mem`

Descrição: A instrução `not`, inverte o valor de cada bit do argumento, ou seja, se um bit tem valor 0 fica com 1 e vice-versa.

Exemplo : $D = !D$

`notw %D`

NEG – Negação por Complemento de dois

`negw reg/mem`

Descrição: A instrução `neg`, faz o valor ficar negativo, ou seja, um valor de x é modificado para -x.

Exemplo: $A = -A$

`negw %A`

AND – Operador E (and)

`andw reg/mem, rem/mem`

Descrição: A instrução `and` executa o operador lógico E (and).

**Exemplo : $D = A \& D$

`andw %A, %D, %D`

ADD - Adição de Inteiros

`addw reg/mem, reg/mem/im*, reg/mem {, reg/mem, reg/mem}`

Descrição: instrução rsub, subtrai o primeiro valor do segundo valor e armazena o resultado no terceiro parâmetro (quarto e quinto opcional).

Exemplo : $S = RAM[A] + D$

`addw (%A), %D, %S`

SUB - Subtração de Inteiros

`subw reg/mem, rem/mem/im*, reg/mem {, reg/mem, reg/mem}`

Descrição: A instrução sub, subtrai o segundo valor do primeiro valor e armazena o resultado no terceiro parâmetro (quarto e quinto opcional).

Exemplo : $A = D - RAM[A]$

`subw %D, (%A), %A`

RSUB - Subtração de Inteiros Reversa

`rsubw reg/mem/im*, rem/mem, reg/mem {, reg/mem, reg/mem}`

Descrição: A instrução rsub, subtrai o segundo valor do primeiro valor e armazena o resultado no terceiro parâmetro (quarto e quinto opcional).

Exemplo : $A = RAM[A] - D$

`rsubw %D, (%A), %A`

INC - Incrementa Inteiro

`incw reg/mem`

Descrição: A instrução inc, adiciona um (1) ao valor do registrador ou memória.

Exemplo : $D = D + 1$

`incw %D`

OR – Operador OU (or)

`orw reg/mem, rem/mem`

Descrição: A instrução or executa o operador lógico Ou (or).

Exemplo: $D = RAM[A] | D$

`orw (%A), %D, %D`

JMP – Jump

`jmp`

Descrição: A instrução jmp executa um desvio, no fluxo de execução, para o endereço armazenado em %A.

`jmp`

JE – Desvia Execução se Igual a Zero

`je reg`

Descrição: A instrução je faz um desvio, no fluxo de execução, para o endereço armazenado em %A, somente se o valor do reg. for igual a zero.

`je %S`

JNE – Desvia Execução se Diferente de Zero

`jne reg`

Descrição: A instrução jne faz um desvio, no fluxo de execução, para o endereço armazenado em %A, somente se o valor do reg. for diferente de zero.

`jne %D`

JG – Desvia Execução se Maior que Zero

`jg reg`

Descrição: A instrução jg desvia, o fluxo de execução, para o endereço armazenado em %A, somente se o valor do reg. for maior que zero.

`jg %S`

JGE – Desvia Execução se Maior Igual a Zero

`jge reg`

Descrição: A instrução jge faz um desvio, no fluxo de execução, para o endereço armazenado em %A, somente se o valor do reg. for maior ou igual a zero.

`jge %S`

JL – Desvia Execução se Menor que Zero

`jl reg`

Descrição: A instrução jl faz desvio, no fluxo de execução, para o endereço armazenado em %A, somente se o valor do reg. for menor que zero.

`jl %S`

JLE – Desvia Execução se Menor Igual a Zero

`jle reg`

Descrição: A instrução jle faz um desvio, no fluxo de execução, para o endereço armazenado em %A, somente se o valor do reg. for menor ou igual a zero.

`jle %D`

NOP – Não faz nada (No Operation)

`nop`

Descrição: A instrução nop não faz nada, usado para pular um ciclo de execução.

`nop`

Operações VM

São suportadas as seguintes operações aritméticas na pilha:

- add
 - executa: $X + Y$
- sub
 - executa: $X - Y$
- neg
 - executa: $-Y$ (complemento de dois)
- eq
 - compara $X == Y$
 - * True : resulta em b“1111111111111111”, 0xFFFF
 - * False: resulta em b“0000000000000000”, 0x0000
- gt
 - compara $X > Y$
 - * True : resulta em b“1111111111111111”, 0xFFFF
 - * False: resulta em b“0000000000000000”, 0x0000
- lt
 - compara $X < Y$
 - * True : resulta em b“1111111111111111”, 0xFFFF
 - * False: resulta em b“0000000000000000”, 0x0000
- and
 - executa: $X \text{ and } Y$ (bit a bit)
- or
 - executa: $X \text{ or } Y$ (bit a bit)
- not
 - executa: not Y (bit a bit)

Função

A seguir definições de funções :

Declaração de função

Uma função é definida pelo keyword **function** seguido do seu **nome** e quantidade de variáveis locais **n** na estrutura a seguir :

function **nome** **n**

Toda função em VM deve possuir um retorno, defindo pelo keyword **return**

Chamada de função

Uma função em VM é chamada pelo keyword: **call** seguido do **nome** da função e da quantidade **m** de parâmetros passados para essa função.

call **nome** **m**

Parâmetros

Os parâmetros de uma função são passados na própria pilha.

Label

Labels são definidas pelo keyword **label** seguido de seu **nome** :

label **nome**

São utilizados para endereçar o código em uma condição de goto.

Goto

Existem dois tipos de GOTO, condicional (**if-goto**) e incondicional (**goto**). No condicional o salto é realizado caso a condição não for Falsa (verifica sempre o último valor da pilha).

goto **nome**

if-goto **nome**