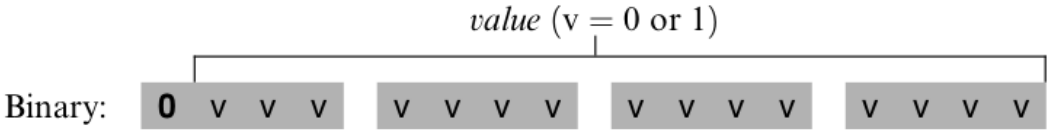
Atividade 4: Linguagem de Máquina

A-instruction

Bit 0 = 0_b

The *A*-instruction is used to set the A register to a 15-bit value:

A-instruction: *@value* // Where *value* is either a non-negative decimal number
// or a symbol referring to such number.



Bits 1 a 15

Número inteiro de 15 bits:

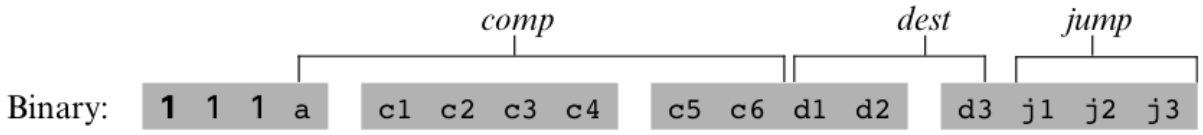
- 0 a 32767, se considerado como *unsigned* (e.g. endereço de memória até 32kb)
- -16384 a 16383, se considerado *signed* (e.g. aritmética pela ALU, complemento de 2)

C-instruction

Bits 0 a 2 = 111_b

C-instruction: *dest=comp;jump*

// Either the *dest* or *jump* fields may be empty.
// If *dest* is empty, the "=" is omitted;
// If *jump* is empty, the ";" is omitted.



Bits 3 a 9

(when a=0) ***comp*** *mnemonic*	c1	c2	c3	c4	c5	c6	(when a=1) ***comp*** *mnemonic*
0	1	0	1	0	1	0	
1	1	1	1	1	1	1	
-1	1	1	1	0	1	0	
D	0	0	1	1	0	0	
A	1	1	0	0	0	0	M
!D	0	0	1	1	0	1	
!A	1	1	0	0	0	1	!M
-D	0	0	1	1	1	1	
-A	1	1	0	0	1	1	-M
D+1	0	1	1	1	1	1	
A+1	1	1	0	1	1	1	M+1
D-1	0	0	1	1	1	0	
A-1	1	1	0	0	1	0	M-1
D+A	0	0	0	0	1	0	D+M
D-A	0	1	0	0	1	1	D-M
A-D	0	0	0	1	1	1	M-D
D&A	0	0	0	0	0	0	D&M
D|A	0	1	0	1	0	1	D|M

Bits 10 a 12

d1	d2	d3	*Mnemonic*	*Destination (where to store the computed value)*
0	0	0	null	The value is not stored anywhere
0	0	1	M	Memory[A] (memory register addressed by A)
0	1	0	D	D register
0	1	1	MD	Memory[A] and D register
1	0	0	A	A register
1	0	1	AM	A register and Memory[A]
1	1	0	AD	A register and D register
1	1	1	AMD	A register, Memory[A], and D register

Bits 13 a 15

j1 ($out < 0$)	j2 ($out = 0$)	j3 ($out > 0$)	Mnemonic	Effect
0	0	0	null	No jump
0	0	1	JGT	If $out > 0$ jump
0	1	0	JEQ	If $out = 0$ jump
0	1	1	JGE	If $out \geq 0$ jump
1	0	0	JLT	If $out < 0$ jump
1	0	1	JNE	If $out \neq 0$ jump
1	1	0	JLE	If $out \leq 0$ jump
1	1	1	JMP	Jump

Tarefa 1

Escreva uma sequência de instruções que simule uma bifurcação de decisão (`if`). Pode ser qualquer ideia do aluno, ou mesmo replicar o exemplo que foi dado em aula:

```
0000000000000000  // A-instruction
1111110000010000  // C-instruction
0000000000000001  // A-instruction
1111000010010000  // C-instruction
0000000000001010  // A-instruction
1110001100000001  // C-instruction
0000000000000010  // A-instruction
1110001100001000  // C-instruction
0000000000001100  // A-instruction
1110101010000111  // C-instruction
0000000000000011  // A-instruction
1110001100001000  // C-instruction
0000000000001100  // A-instruction
1110101010000111  // C-instruction
```

Descreva os passos do programa, instrução a instrução.

Tarefa 2

Escreva uma sequência de instruções que simule uma iteração (`for`). Pode ser qualquer ideia do aluno, por mais simples que seja.

Dica: use um dos endereços da memória para armazenar a quantidade de iterações a serem executadas e, a cada iteração, subtraia 1 desse total. Enquanto não chegar a zero, volte à primeira instrução do bloco interior do laço. Quando chegar a 0, force um *jump* para fora do bloco do laço.

Ideia 1: Para aqueles que não tem uma ideia clara de qual aplicação pode fazer com o laço, use um outro endereço de memória e carregue-o com “0”. A cada iteração, some 1 a esse endereço. O resultado final será visto como a transposição de endereço do número de iterações executadas. Exemplo: no endereço 0, escrevemos 5; no endereço 1, escrevemos 0. Ao final do código, o endereço 0 deve conter 0 e o endereço 1 deve conter 5.

Ideia 2: Implemente um algoritmo de multiplicação de dois números na memória.

Descreva os passos do programa, instrução a instrução.

Referências para o trabalho

O site http://www.nand2tetris.org contém o simulador de CPU a ser usado no trabalho. Encontra-se em https://www.nand2tetris.org/software no link indicado aqui:

Download the Nand2tetris Software Suite Version 2.6 (about 730K).

Usaremos o simulador **`CPUEmulator.sh`** (linux/mac) ou o **`CPUEmulator.bat`** (windows) para rodar os códigos de máquina. Assista a aula novamente para dicas de instalação e de utilização do mesmo.

Na dúvida, o livro ***The Elements of Computing Systems: Building a Modern Computer from First Principles*** de Noam Nisan e Shimon Schocken será nosso guia para essa atividade, em especial o capítulo 3 (*Sequential Logic*). Os padrões de implementação são os que eles propõem para chegarmos na construção do *Hack computer system*.

Entrega

Os alunos deverão dar upload na área de *assignments* do Teams de seus arquivos ".hack" gerados assim como o ".pdf" contendo a descrição dos códigos, compactados em um único arquivo (.zip, .7z, .tar.xz, …), até a **data definida no *assignment***.