Relatório

Nomes dos integrantes do grupo:
Giovanna Bolsoni, Barbara Fernandes e Milene Lopes.

Estratégia utilizada pelo algoritmo:
Fizemos dois loopings, um para a multiplicação e outro “principal” que pega o valor do expoente, tira uma unidade dele (para não fazer multiplicações a mais, porque a primeira já conta como uma), e conta as vezes que o looping de multiplicação irá rodar. Além disso, fizemos “comparações”, utilizando o JZ (que verifica se o acumulador está zerado e pula para a posição descrita posteriormente), onde caso o expoente seja zero, o neander pula direto para uma posição que coloca o valor de 1 na posição 130 (resultado) e já encerra o programa. O mesmo para números onde o expoente é um (quando se tira uma unidade para não fazer multiplicações excessivas e o expoente é 1, o valor zera, então o JZ pula para as instruções nas linhas finais, pegando o valor da própria base, colocando no resultado e finalizando o programa). Também, utilizamos uma auxiliar para guardar os resultados das multiplicação, para não somar os resultados na hora de fazê-los, e assim, no final colocar o valor dela na posição de resultado.

Explicação do código, desde o começo:

Posições de [0,3]: Começa carregando o valor da posição 129 (expoente) no acumulador e verifica se é zero ou não, porque se for, ele pula para a posição 68, onde o programa coloca 1 na posição 130, que é a posição de resultado e finaliza o programa.

Posições de [4,9]: Caso não seja zero, ele prossegue tirando uma unidade do expoente, para não exceder a quantidade de multiplicações e armazena na posição do iterador principal (que seria a quantidade de loopings de multiplicação que ele irá fazer).

Posições de [10,13]: Depois ele verifica se esse expoente é zero, porque se for, significa que anteriormente era 1 (antes de adicionar -1 e tirar uma unidade), portanto é o resultado é o valor da própria base, então pula para a posição 74, onde o programa carrega o valor da base no acumulador e coloca na posição 130 (resultado) e finaliza o programa.

Posições de [14,19]: Carrega o valor da base, coloca no número de iterações de multiplicação, e depois carrega o valor dessas iterações no acumulador, para entrar no looping da primeira multiplicação.

Posições de [20,35]: Looping da multiplicação, adicionando o valor da base, quantas vezes o iterador de multiplicações estiver apontando, e decrementa ele de um em um, até zerar, e depois disso, vai para a posição 36.

Posições de [36,45]: Coloca o valor do resultado (130), em uma posição auxiliar (135) e decrementa uma posição do iterador principal (133).

Posições de [46,53]: Coloca o valor do resultado (130), na posição de iteração de multiplicação (131) e zera a posição de resultado para poder começar a multiplicação novamente (se for preciso).

Posições de [54,61]: Carrega a posição da iteração principal e se não for zero, carrega o valor da posição de iteração de multiplicações no acumulador e volta para o looping de multiplicações para fazer o caminho todo de novo. Agora, se for zero, ele pula para a posição 62, coloca o valor da auxiliar na posição de resultado e pula para o fim do programa.

Posições de [62,67]: Carrega a auxiliar no acumulador e coloca na posição de resultado e pula para finalizar o programa.

Posições de [68,73]: Carrega o valor da posição 134 (que é 1), coloca no resultado (caso alternativo, onde o número é elevado a 0), e pula para finalizar o programa.

Posições de [74,78]: Carrega o valor da posição 128 (que é a base inserida), coloca no resultado (caso alternativo, onde o número é elevado a 1, ou seja, é igual a ele mesmo), e finaliza o programa.

Rotina Desenvolvida:

```
LDA 129 -
JZ  68  -
LDA 129  -
ADD 132  -
STA 133  -
LDA 133  -
JZ  74  -
LDA 128  -
STA 131  -
LDA 131 -
JZ 36  -
LDA 130  -
ADD 128  -
STA 130  -
LDA 131  -
ADD 132  -
STA 131 -
JMP 20  -
LDA 130  -
STA 135  -
LDA 133  -
ADD 132  -
STA 133  -
LDA 130  -
STA 131  -
LDA 136  -
```

STA 130 -
LDA 133 -
JZ 62 -
LDA 131 -
JMP 20 -
LDA 135 -
STA 130 -
JMP 78 -
LDA 128 -
STA 130 -
HLT -

Casos de teste utilizados:
Fizemos na mão testes com: 2 elevado a 2, 2 elevado a 0 e 2 elevado a 1, e também no site passado pelo professor, os seguintes testes (com números maiores):

Teste com 5 elevado a 0:
(arquivo .hexmem)

034e445220008100a000440020008100300084001000850020008500a000
4a00200080001000830020008300a000240020008200300080001000820 0
2000830030008400100083008000140020008200100087002000850030 00
840010008500200082001000830020008800100082002000850 0a0003e00
2000830080001400200087001000820080004e00200086001000820080 00
4e00200080001000820 0f000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000500000001000000ff00
000001000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000

Teste com 3 elevado a 4:
(arquivo .hexmem)

034e445220008100a000440020008100300084001000850020008500a000
4a002000800010008300200083 00a000240020008200300080001000 8200
200083003000840010008300800014002000820010008700200085003000
840010008500200082001000830020008800100082002000 8500a0003e00
200083008000140020008700100082008000 4e002000860010008 2008000
4e002000800010008200f000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000300040051005100ff00
000001005100000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000

Dados auxiliares utilizados na memória de dados:
Utilizamos os seguintes dados:
Posição 128 (chamamos de A, que é a base): Valores utilizados nos testes.
Posição 129 (chamamos de n, que é o expoente): Valores utilizados nos testes.
Posição 130 (chamamos de Resultado): Inicializada vazia, com 0.
Posição 131 (chamamos de Iterador de Multiplicação): Inicializada vazia, com 0.
Posição 132 (usamos para guardar o número -1 e utilizá-lo nos decrementos necessários): Inicializada com -1.
Posição 133 (chamamos de Iterador de Multiplicação): Inicializada vazia, com 0.
Posição 134 (usamos para guardar o número 1 e utilizá-lo quando o expoente for zero): Inicializada com 1.
Posição 135 (chamamos de Auxiliar): Inicializada vazia, com 0.
Posição 136 (usamos para zerar a posição resultado): Inicializada vazia, com 0.

Memória de Processamento
de Instruções

0	LDA	30	ADD	60	JMP
1	129	31	132	61	20
2	JZ	32	STA	62	LDA
3	68	33	131	63	135
4	LDA	34	JMP	64	STA
5	129	35	20	65	130
6	ADD	36	LDA	66	JMP
7	132	37	130	67	78
8	STA	38	STA	68	LDA
9	133	39	135	69	134
10	LDA	40	LDA	70	STA
11	133	41	133	71	130
12	JZ	42	ADD	72	JMP
13	74	43	132	73	78
14	LDA	44	STA	74	LDA
15	128	45	133	75	128
16	STA	46	LDA	76	STA
17	131	47	130	77	130
18	LDA	48	STA	78	HLT
19	131	49	131	79	
20	JZ	50	LDA	80	
21	36	51	136	81	
22	LDA	52	STA	82	
23	130	53	130	83	
24	ADD	54	LDA	84	
25	128	55	133	85	
26	STA	56	JZ	86	
27	130	57	62	87	
28	LDA	58	LDA	88	
29	131	59	131	89	

Memória de Dados

"A" 128	2
"n" 129	2
Resultado 130	
Iterador Multi 131	
132	-1
Iterador principal 133	
134	1
Auxiliar 135	
136	0
137	
138	