Exercício avaliativo 1

Instituto Federal de Educação, Ciência e Tecnologia da Paraíba (IFPB)
Disciplina: Microprocessadores e microcontroladores
Professor: Fagner de Araujo Pereira
Aluno (a): _____

Exercício avaliativo 1 (Peso 100 pontos) - Gabarito

1. A figura abaixo mostra os campos de bits de uma instrução de um processador genérico **com arquitetura de 32 bits**. Considerando que sua ULA é capaz de executar 320 operações distintas com até 2 operandos e salvar o resultado no destino, todos presentes em um banco de registradores nomeados de R0 a R31, determine o que se pede:

opcode	*endereço dos operandos*		
código da operação	*operando A*	*operando B*	*destino*

a) Quantos bits possui o campo código da operação?
$n = [log_2 320] = 9$ bits

b) Quantos bits possui o campo de endereço dos operandos?
Uma vez que os registradores são nomeados de R0 a R31, temos 32 registradores no banco. Para endereçar cada registrador são necessários 5 bits. Logo, o campo de endereço dos operandos, que é formado pelo endereço de 3 registradores, possui 15 bits.

c) Qual o comprimento da instrução?
Comprimento = 9 + 15 = 24 bits

d) Qual a capacidade de endereçamento direto de uma instrução LOAD, com o mesmo comprimento da instrução apresentada?
No endereçamento direto de uma instrução LOAD, devem ser indicados o registrador de destino e o endereço de memória onde o dado está armazenado. Como o campo de endereços possui 15 bits e 5 bits devem ser usados para indicar o registrador de destino, sobram 10 bits para indicar o endereço de memória, o que implica num espaço de endereçamento de $2^{10} = 1024$ bytes, ou 1kB.

e) Qual a capacidade de endereçamento indireto via registrador?
No endereçamento indireto via registrador, o endereço é obtido a partir do conteúdo de um registrador. Como a arquitetura é de 32 bits, o endereço possui 32 bits, o que implica num espaço de endereçamento de $2^{32} = 4294967296$ bytes, ou 4GB.

2. Considere um processador no qual há instruções de um operando cujo destino é sempre um registrador conhecido como ACUMULADOR. A instrução **LDR** carrega o operando no acumulador; **MUL** multiplica o operando pelo valor que está no acumulador; **ADD** realiza a soma do operando com o valor do acumulador; **STR** armazena o valor do acumulador no endereço fornecido pelo operando. Com base nessas instruções, um engenheiro escreveu o seguinte código:

```
LDR X
MUL #3
ADD #4
MUL #2
STR Y
```

Y = (3X+4)*2 = 6X+8

Assinale a alternativa que corresponde à operação implementada pelo código descrito:

a) Y = 3X + 8
b) Y = 12X + 2
c) Y = 12X+8
d) Y = 6X+8
e) Y = 6X + 12

3. O trecho de código mostrado abaixo é a transcrição em linguagem Assembly de uma sub-rotina de atraso (*delay*) obtida a partir do processo de compilação para um processador com arquitetura de **16 bits**.

```
delay: MOV R0, 310          //move 310 para o registrador R0
       CLR R1               //limpa o registrador R1
step:  DJNZ R1, step        //decrementa R1 e desvia para step se for diferente de zero
       DJNZ R0, step        //decrementa R0 e desvia para step se for diferente de zero
       RET                  //retorna da sub-rotina
```

Considere que o clock desse processador é de 100 MHz, e que cada instrução é executada em um ciclo de clock. Nessas condições, a sub-rotina gera um atraso de quanto tempo?

Primeiro, vamos contabilizar quantos ciclos de clock o programa necessita. Nas duas primeiras linhas, temos:

```
delay: MOV R0, 310    (1 ciclo)
       CLR R1         (1 ciclo)
```

Quando ocorre a primeira execução da linha `DJNZ R1, step`, ocorrerá um underflow em R1, fazendo com que todos os seus bits sejam levados a 1. O teste então verificará que R1 ≠ 0 e o programa desviará para step, que é a própria linha que ele está atualmente. Com isso, essa linha será executada repetidamente, até que R1 seja zero novamente. Isso representa 2^{16} execuções, uma vez que o registrador é de 16 bits. Logo:

```
step:  DJNZ R1, step  (2^16 ciclos)
```

A linha seguinte, `DJNZ R0, step`, é executada 1 vez e, até que R0 decresça de 310 até zero, o programa desviará para `step`, repetindo o processo anterior. Isso significa que esse segundo desvio ocorrerá 310 vezes, fazendo com que, em cada vez, o processo anterior seja repetido. Logo:

```
DJNZ R0, step  (310 ciclos)
```

E a quantidade de ciclos gastos nessas duas linhas é:

$$\{2^{16}\} * \{310\} = 20316160 \text{ ciclos}$$

Finalmente, na última linha, temos:

```
RET (1 ciclo)
```

Assim, a quantidade total de ciclos de clock é:

$$Ciclos\_de\_clock_{totais} = 1 + 1 + 20316160 + 1 = 20316163\ ciclos$$

Dessa forma, o tempo total de execução da sub-rotina é a quantidade de ciclos de clock multiplicada pelo tempo de cada pulso de clock, que é igual ao período de 100 MHz:

$$tempo = 20316163 * \frac{1}{100M} \cong 203{,}16\ ms$$

4. No contexto de processadores, a arquitetura ARM evoluiu de um determinado princípio de projeto e possui algumas aplicações marcantes. Assinale a alternativa que contém esse princípio de projeto e a aplicação marcante.

a) CISC; sistemas distribuídos
b) RISC; sistemas embarcados
c) RISC; sistemas operacionais
d) CISC; sistemas embarcados
e) RISC; sistemas distribuídos

Exercício avaliativo 1

5. Abaixo, são mostrados os conteúdos do banco de registradores e da memória de dados de um processador genérico de 32 bits em um determinado instante.

Conteúdo inicial do banco de registradores

Registrador	Conteúdo inicial	Conteúdo final
R7 (PC)	1230	1243 (1282)
R6 (LR)	1150	1150
R5 (SP)	702	699
R4	250	702
R3	250	75
R2	0	250
R1	100	1010
R0	815	75

Memória RAM

Endereço	Conteúdo inicial	Conteúdo final
703	-200	-200
702	80	1150
701	-40	-40
700	-30	-30
699	75	1010
698	500	500
697	-170	-170
696	500	500

A pilha dessa arquitetura é ascendente e as operações funcionam da seguinte forma: para inserir um item, primeiro o ponteiro de pilha é incrementado e depois o dado é inserido; para retirar um dado, primeiro o dado é retirado e depois o ponteiro é decrementado.
Sabendo dessas informações, preencha a terceira coluna das tabelas apresentadas com o conteúdo final após a execução do código abaixo, localizado em posições sequenciais na memória de programa, que é iniciado nesse instante.

```
start: MOV R2, R3          //transferência de dados
       SHL R1, R0, #2      //deslocamento de #n bits
       MOV R4, #3          //armazenamento de #constante
       MUL R3, R3, R4      //multiplicação
       POP R0              //operação de pilha

test:  BRZ R4, end         //desvia para o endereço alvo se o argumento é zero
       SUB R1, R1, R3      //subtração
       POP R0              //operação de pilha
       DEC R4              //decremento
       BR test             //desvio incondicional

end:   ADD R3, R0, R4      //adição
       PUSH R1             //operação de pilha
       XOR R4, R1, #332    //operação lógica com #constante
       STR [R4], R6        //acesso à memória
```

(nota 1: todos os valores referenciados são representados no formato decimal)
(nota 2: o primeiro registrador de uma instrução de transferência ou de processamento de dados é o destino e os demais são os operandos)
(nota 3: As palavras "start", "test" e "end" no código acima são referências de endereços na memória de programa)