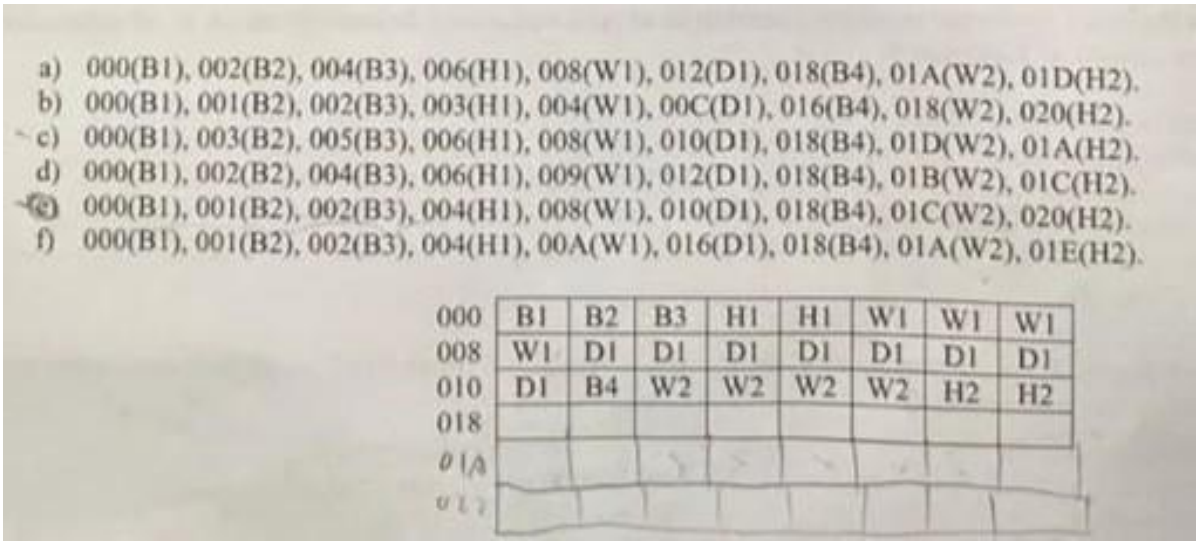
Primeiro Miniteste

1- Assinale a opção que representa o conteúdo da memória de dados referente ao trecho de código abaixo utilizando o tipo de ordenação de bytes Little Endian

```
int main(){
   struct{
      int x;
      char y;
   } data;
   data.x □ 0x61626364
   data.y = '\0';
}
```

	0000	0001	0002	0003	0004
a	61	62	63	64	00
b	00	61	62	63	64
c	64	63	62	61	00
d	00	64	63	62	61

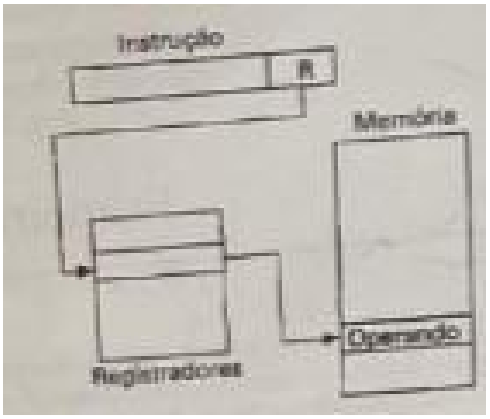
2- Indique a opção de endereços capaz de alinhar todos os objetos apresentados no mapa de memória abaixo:



Resposta correta e

3- Selecione a opção que representa o modo de endereçamento ilustrado pela figura:

- Imediato
- Direto
- Registrador
- Indireto
- Deslocamento
- Indexado



d. Indireto

Endereçamento indireto é um modo de acessar dados em memória onde a instrução não contém diretamente o endereço do dado, mas sim um endereço que aponta para outro endereço na memória onde o dado efetivamente está armazenado.

Ou seja, a instrução contém um ponteiro para o endereço que possui o valor desejado. Para obter o dado, o processador precisa fazer dois acessos à memória: primeiro para buscar o endereço que está armazenado, e depois para buscar o dado no endereço encontrado.

4- Indique a opção que ilustra a ação de movimentação que utiliza o modo de endereçamento apresentado no Exercício 3:

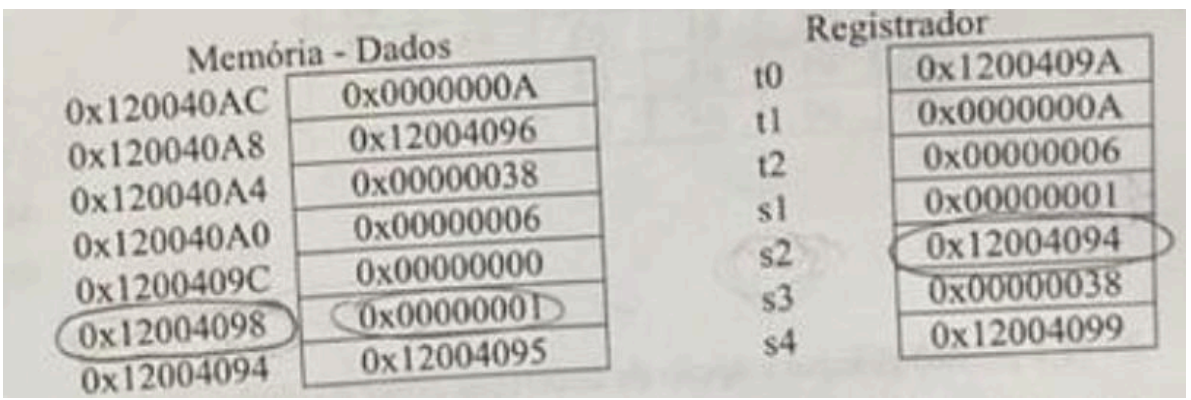
- Load (@R1)
- Load (R1)
- Load #10
- Load (10)
- Load (R1 + (100))

Sintaxe do load

LD A, (BC)

Carrega o registrador A com o valor da memória no endereço apontado pelo registrador BC.

5- Utilizando o mapa de memória abaixo, indique o conteúdo do registrador t0 após a execução da instrução RISC-V `lw t0, 8(s2)`:



a) 0x12004099
b) 0x1200409A
c) 0x00000006
d) 0x00000038
e) 0x00000001
f) 0x00000000
g) 0x12004095

Sintaxe do lw:

lw registrador_destino, offset (registrador_endereço_base)

O registrador de endereço base é s2 que tem o endereço 0x12004094 e devemos somar a ele o offset 8. Logo temos:
0x12004094 + 0x8 = 0x120049C
E neste endereço está armazenado o valor 0x00000000 que será então armazenado no registrador t0

6- Indique a opção que contém o conteúdo do registrador `s4` após a execução da instrução `slt s4, s1, s3` utilizando o mapa de registradores do exercício 5:

a) 0x00000001

b) 0xFFFFFFFF

c) 0x00000038

d) 0x00000000

e) 0x0000000F

f) 0x12004099

Sintaxe do slt

slt rd, rs1, rs2 (Set Less Than):

Compara os valores com sinal de rs1 e rs2

If rs1 < rs2, rd é set = 1.

Otherwise, rd é set = 0.

Olhando no mapa de registradores s1 = 0x00000001 e s3 = 0x00000038

0x00000001 < 0x00000038 = true = 1

Logo, s4 = 0x00000001

7- Supondo que o registrador PC aponte para a instrução `0x00400004`, apresente o valor desse registrador após a execução da instrução apontada.

Memória de Instrução	Instrução
0x00400000	add s0, s1, s2
0x00400004	jal x0, 0x0000000C
0x00400008	sub s0, s1, s2

0x0040000C	lw t0, 4(s2)
0x00400010	slt s4, s1, s3

a) 0x00400000

b) 0x00400004

c) 0x00400008

d) 0x0040000C

e) 0x00400010

Devemos olhar a memória de instrução e relacionar o endereço da instrução com a instrução em assembly. Com isso temos que `0x00400004` = jal x0, 0x0000000C.

Sintaxe do jal

jal rd, offset

rd: registrador de destino que armazenará o endereço retornado

offset: imediato que será somado a PC para realizar o jump

0x0000000C + 0x00400004 = 0x00400010

Obs: 0x0000000C = 12 em decimal, logo podemos contar 3 instruções (cada instrução vale 4) depois do jal

8- Apresente o código de máquina em hexadecimal da instrução apontada pelo PC descrita no exercício 7.

opcode JAL = 1101111(0x6F)
rd = x0 = 0x0000
imm = 0x0000000C: 00000000000000001100

20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0

J- type

imm 20, 10:1, 11, 19:12 (20 bits)	rd (5 bits)	op (7 bits)
0000_0000_1100_0000_0000	0000_0	110_1111

0000	0000	1100	0000	0000	0000	0110	1111
0	**0**	**C**	**0**	**0**	**0**	**6**	**F**

0x00C0006F

Bkpt	Address	Code	Basic	Source
	0x00400000	0x00c0006f	jal x0,0x0000000c	5: jal x0, destino # pular para o label ..
	0x00400004	0x00100513	addi x10,x0,1	8: li a0, 1 # (exemplo)
	0x00400008	0x00100513	addi x10,x0,1	9: li a0, 1 # (exemplo)
	0x0040000c	0x00200513	addi x10,x0,2	12: li a0, 2 # código executado ap..
	0x00400010	0x00a00893	addi x17,x0,10	13: li a7, 10 # código syscall para..
	0x00400014	0x00000073	ecall	14: ecall

9- As instruções listadas abaixo permitem:

```
addi sp, sp, -4

sw s0, 0(sp)
```

a) Remover o topo da pilha e armazená-lo em s0;

b) Inserir o conteúdo do registrador s0 no topo da pilha;

c) Subtrair 4 do valor contido no topo da pilha e armazená-lo em s0;

d) Realizar uma operação de swap (troca) entre s0 e sp;

e) Multiplicar o conteúdo de sp por 0 e armazená-lo em s0;

b. Inserir o conteúdo do registrador s0 no topo da pilha;

addi sp, sp, -4 -> A pilha cresce para baixo(diminuindo o endereço), para abrir espaço na pilha, precisa diminuir o SP

sw s0, 0(sp) -> salva o valor de s0 na nova posição no topo da pilha

10- O trecho de código abaixo utiliza o conceito de sub-rotina para uma tarefa. Indique a alternativa que substitui respectivamente as linhas xxxxx e yyyyy do código apresentado.

```
li a0,10
li a1,21
xxxxx
li a7, 10
ecall
PROC1:
  add a0,a0,a1
  yyyyy
```

a. j PROC1 e jr a1

b. beq a1, a0, PROC1 e jal ra

c. jal PROC1 e jr ra

d. jal PROC1 e jal ra

e. bne a1, a0, PROC1 e jal ra

f. ecall PROC1 e jal ra

c. jal PROC1 e jr ra

O código chama a sub-rotina com `jal ra, PROC1`, que além de desviar a execução, salva em `ra` o endereço da próxima instrução (`li a7,10`). Na sub-rotina, a soma `add a0,a0,a1` é realizada e, em seguida, o comando `jr ra` faz o retorno para o ponto salvo, garantindo que o programa continue no fluxo correto e finalize com o `ecall`.

11- A sequência de bits abaixo representa uma instrução RISC-V de 32 bits R-Type. Indique a opção que apresenta respectivamente o registrador destino, o primeiro operando e o segundo operando presente na instrução.

0100000_00101_01110_000_01010_0110011

a) a0, a4 e t0

b) t0, s2 e s1

c) s0, t1 e s2

d) s3, s4 e s5

e) a0, t1 e t2

f) t1, t2 e s2

g) t8, s7 e s6

a. a0, a4 e t0

rd = registrador destino -> 01010 = x10 = a0

rs1 = operando 1 -> 01110 = x14 = a4

rs2 = operando 2 -> 00101 = x5 = t0