UFFS – Universidade Federal da Fronteira Sul
Curso de Ciência da Computação
Disciplina: Organização de Computadores
Professor: Luciano L. Caimi **Data: 14/05/2025**

Aluno: Gideon G. Müller **Nota:** ~~8,2~~ 8,9

1,0

1. **(1.0)** Existem 2 modelos de acesso à memória, conhecidos como modelo de acesso tipo Von Neumann e modelo de acesso tipo Harward. a) Descreva cada um dos modelos de acesso a memória. b) Considerando o ciclo de instrução de máquina, explique porque o modelo de acesso tipo Von Neumann representa um gargalo.

2,0 ~~1,5~~ 0,8 Pode fazer junto

2. Utilizando o conjunto de instruções do RISC-V, faça o que se pede:
a (2.0) - Implemente a função `mul_with_sum` que recebe em a0 e em a1 dois valores inteiros. A função deve retornar em a0 o valor da multiplicação dos valores enviados para a função. A multiplicação deve utilizar o algoritmo das somas sucessivas, por exemplo: -5 x 3 = -5 + -5 + -5 = -15. A função deve tratar números positivos e negativos tanto em a0 quanto em a1.
b (1.0) - Implemente um programa que apresente as mensagens no console, leia dois valores inteiros no teclado, chame a função `mul_with_sum` e mostre o resultado da função no console.

1,5

3. **(1.5)** Sabendo que S é um endereço de memória e considerando o conjunto de instruções, o programa apresentado e os valores presentes na pilha. Apresente os valores presentes na pilha ao final da execução das linhas 5, 9, 11 e 12.

Conjunto de Instruções
POP op; (op) ← topo
PUSHI op ; topo ← op
PUSH op; topo ← (op)
POW ; topo ← topo ^ topo$_{-1}$
ADD ; topo ← topo + topo$_{-1}$
SUB ; topo ← topo - topo$_{-1}$
MUL ; topo ← topo * topo$_{-1}$
DIV ; topo ← topo / topo$_{-1}$

linha	Programa
1	PUSHI 4
2	POW
3	ADD
4	MUL
5	DIV
6	PUSHI 4
7	ADD
8	DIV
9	ADD
10	POW
11	ADD
12	POP S

Pilha	
topo →	2
	-10
	2
	6
	-3
	5
	2
	-1
	-4

1,4

4. **(1.5)** Para cada uma das instruções do RISC-V codificadas em hexadecimal mostradas abaixo, apresente a instrução Assembly completa (exemplo: add s0, s1, s2) e classifique a instrução quanto ao formato (R, I, B, etc), classe de instrução (aritmética, lógica, desvio condicional, etc) e o modo de endereçamento (registrador, imediato, relativo ao PC, etc).

Código	Instrução	Formato	Classe	Modo de endereçamento
0x00B50313	addi t1, a0, 11	I	Aritmético	Imediato
0x0122A023	sw s2, 0(t0)	S	LOAD/STORE	Base + Deslocamento
0x0240006F	JAL zero, 130	UJ	Branchings	Base + Deslocamento

Acesso à memória

JAL zero, 36

5. **(3.0)** Considere o programa usando o ISA RISC-V e o valor atual dos registradores conforme mostrado abaixo. Sabendo que o valor atual de PC = 0x00004008, apresente os valores presentes nas posições de A até K nos dois próximos ciclos de instrução, do processador RISC-V monociclo (figura abaixo):

Programa

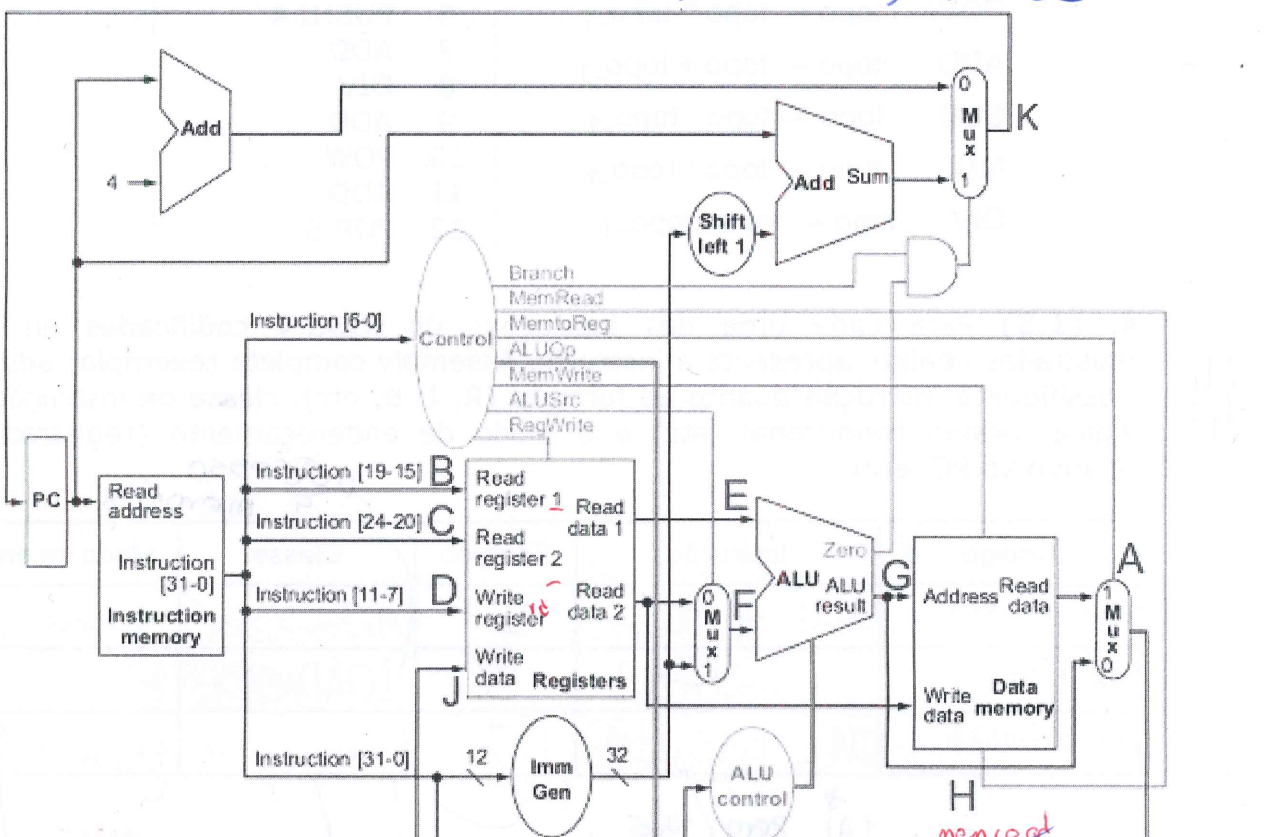
```
        .data   0x00001000
vet:    .word    -3, 6, 5, 0, -7

        .text   0x00004000
main:
     addi t0, zero, 5
     li   a1, 0
laco:
     lw   a3, 4 (a2)
     beq  t0, a3, out_laco
     addi a0, a0, -4
     addi a1, a1, 1
     addi a2, a2, 4
     j laco
out_laco:
     mv   a0, a1
     ret
```

Banco de registradores

zero	0x00000000
a0	0x00000006
a1	0x00000001
a2	0x00001004
a3	0x00000006
t0	0x00000005

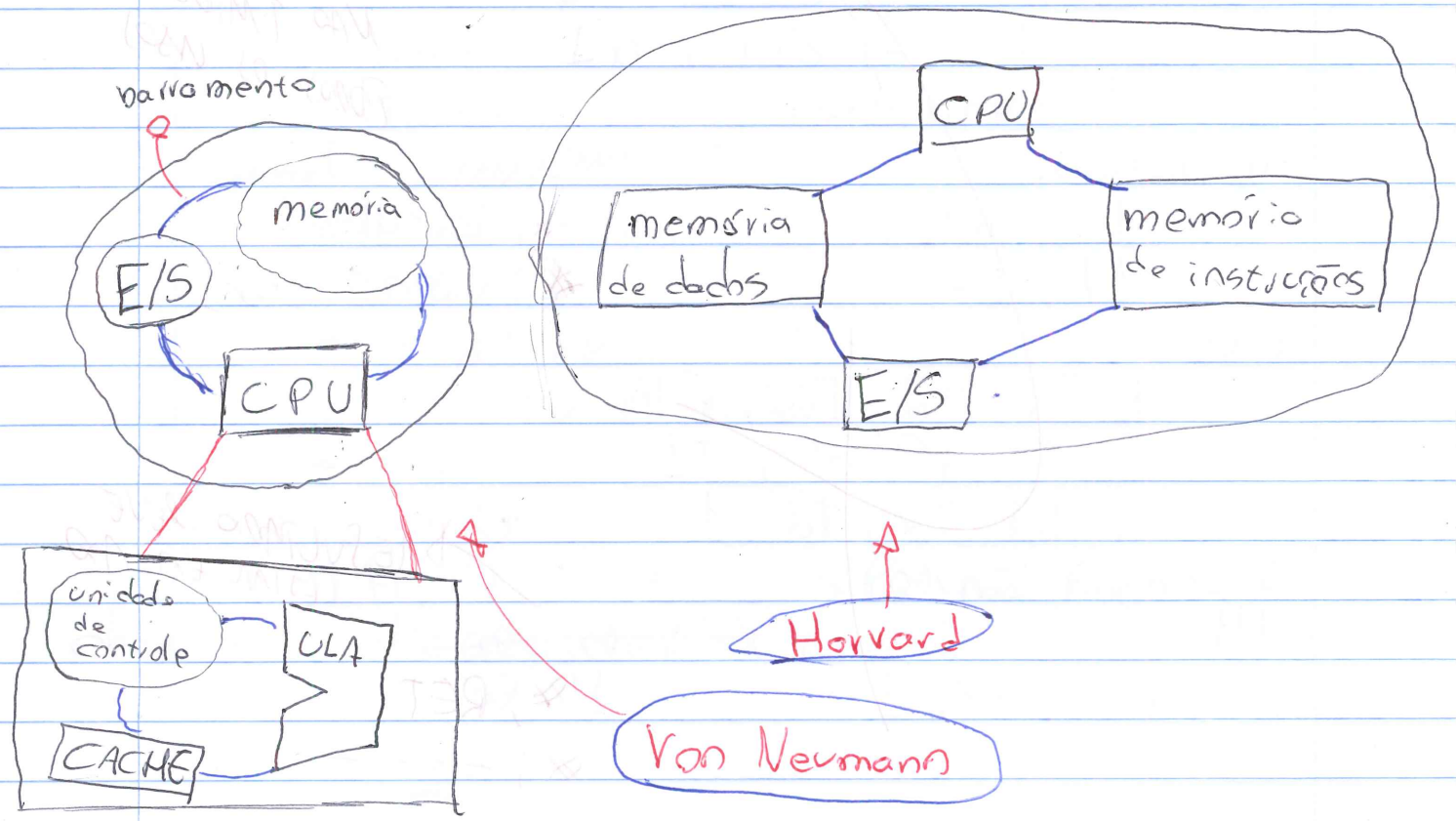
posição	PC = 0x00004008	Próximo PC
A		
B		
C		
D		
E		
F		
G		
H		
J		
K		



Erickson G. Muller

1-) A arquitetura de Von Neumann introduziu o conceito de Programa armazenado, Sua unidade de memória armazena tanto dados quanto instruções e se comunica com a CPU e os dispositivos de entrada e saída por meio de um barramento único, daí que se visualiza o Gargalo de Von Neumann, nesse modelo de acesso à memória, o ciclo de instruções só consegue fazer um acesso à memória por vez, buscando ou um dado ou uma instrução. Para administrar esse problema de desempenho, a arquitetura faz uso da memória cache.

A arquitetura Harvard separa essa memória em memória de dados e memória de instruções.



alterações para letra b

2) a) mul_with_sum:

```
LI a0, 3          # ; MV a0, a2
LI a1, -5         # ; MV a2, a3
bgt a0, zero, primeiro_positivo
bgt a1, zero, segundo_positivo
j nenhum_positivo
primeiro_positivo:
    MV T0, a0
    MV T1, a1
    j Parametros
segundo_positivo:
    MV T0, a1
    MV T1, a0
    j Parametros
nenhum_positivo:
    SUB T0, zero, a0
    SUB T1, zero, a1
    j Parametros
Parametros:
    LI a0, 0      # ; Resultado
    LI t2, 0      # ; contador
LOOP:
    BEQ T2, T0, FIM
    ADD a0, a0, T1
    ADDI t2, t2, 1
    J LOOP
FIM:              # ; Para a questão B isso será substituído
    LI a7, 10     # ; RET
    ecall         # ; ----
```

NÃO TMBM POIS OS USOS

RESULTADO DEVE ESTAR EM A0

2-) b)

```
.data
LATA: .strings "Digite um número: "

.text
main:
```

* as alterações da função foram comentadas
* em vermelho na ~~função~~ letra a.

```
NUM_1:
    JAL print_com
    LI a7, 5
    ecall
    MV a2, a0
NUM_2:
    JAL print_com
    LI a7, 5
    ecall
    MV a3, a0
chamada: PARAMETROS DEVEM ESTAR EM A0 E A1
    JAL mul_with_sum
    LI a7, 1
    ecall * ; Imprime resultado
    LI a7, 10 * encerra
    ecall
print_com: a0
    LA a4, LATA
    ~~LW a0, 0(a4)~~ X  > PQ não funciona?
    LI a7 4
    ecall
    RET
```

3-)

LINHA 5	LINHA 9	LINHA 11	LINHA 12
PILHA	PILHA	PILHA	PILHA
2 ← TOPO	3 ← TOPO	8 ← TOPO	-4 ← TOPO
-3	2	-4	
5	-1		
2	-4		
-1			
-4			

4-)

			rs1	f3	rd/imm	OP	
I =	0000 0000	1011	0101	0000	0011	0001	0011 ✓
II =	0000 0001	0010	0010	1010	0000	0010	0011 ✓
III =	0000 0010	0100	0000	0001	0000	0110	1111 ✓

0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 = 130 → 36

0 rs1 1 f3 rd

5-) LW a3, 4(a2)

0000 0000 0100 | 01100 | 010 | 01101 | 0000011

BEQ t0, a3, outloop

rs1 rs2 imm

7 5 5 3 5 7

0000000 1 01101 00101 000 01000 1100011

rs2 rs1 f3

2e

1	2	3	4	5	6	7	8	9	10	11	12
12	11	10	9	8	7	6	5	4	3	2	1
11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	1	0	0