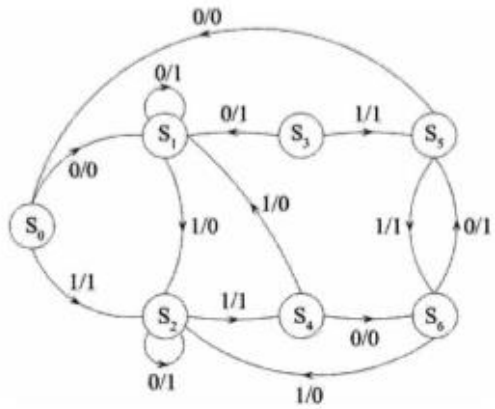
Departamento de Sistemas e Computação	**Universidade Federal de Campina Grande** **Departamento de Sistemas e Computação** **Disciplina: *Organização e Arquitetura de Computadores I*** **Prof. *Joseana Macêdo Fechine Régis de Araújo*** **2º Exercício de Avaliação (Miniprova 1)** **DATA: 09/09/2016**	
Matrícula	**Nome**	**Nota**

1. Responda as questões a seguir.

3,5

1.1 O diagrama abaixo representa uma máquina de estados síncrona de Mealy, com um bit de entrada e um bit de saída. Cada par X/Y indica uma possível entrada X e a correspondente saída Y, na condição representada. Os sete estados possíveis, por sua vez, são representados por S0, S1, S2, S3, S4, S5 e S6. Considerando que a máquina acima descrita esteja inicialmente no estado S0 e que seja aplicada na entrada a sequência de 0-0-1-0-0-1-0-0-1-0-0-1 (um bit para cada ciclo de relógio), assinale a opção que apresenta a sequência correta de bits de saída correspondente a essa sequência de entrada.



a) 0-1-1-0-1-1-0-1-1-0-1-1
b) 1-0-0-1-0-0-1-0-0-1-0-0
c) 1-1-0-1-1-0-1-1-0-1-1-0
d) 0-1-0-1-1-1-0-1-1-1-0-1
e) 0-1-0-1-0-1-1-0-0-1-1-1

1.2 Considere as afirmações a seguir.

I. Uma característica típica de uma arquitetura RISC é o conjunto reduzido de registradores.
II. Basicamente, “Little Endian” significa que o byte de menor ordem do número é armazenado na memória nos menores endereços, e os de maior ordem nos maiores endereços. “Big Endian” significa que os bytes de maior ordem de um número serão armazenados nos menores endereços, e os de menor ordem nos maiores endereços.
III. A arquitetura RISC-V é uma arquitetura CISC que apresenta uma série de registradores de uso geral, além de ser caracterizada por apresentar uma operação elementar por ciclo de máquina.
IV. Se um computador é de 32 bits e outro de 64 bits, significa que esses computadores adotam células de memória com tamanho 32 e 64 bits, respectivamente.

Está(ão) correta(s) apenas a(s) afirmativa(s):
a) I, II e III. b) I, II e IV. c) II. d) IV.

1.3 Apesar de todo o desenvolvimento, a construção de computadores e processadores continua, basicamente, seguindo a arquitetura clássica de von Neumann. As exceções a essa regra encontram-se em computadores de propósitos específicos e nos desenvolvidos em centros de pesquisa. Assinale a opção em que estão corretamente apresentadas características da operação básica de um processador clássico.

a) Instruções e dados estão em uma memória física única; um programa é constituído de uma sequência de instruções de máquina; uma instrução é lida da memória de acordo com a ordem dessa sequência e, quando é executada, passa-se, então, para a próxima instrução na sequência.
b) Instruções e dados estão em memórias físicas distintas; um programa é constituído de um conjunto de instruções de máquina; uma instrução é lida da memória quando o seu operando-destino necessita ser recalculado; essa instrução é executada e o resultado é escrito no operando de destino, passando-se, então, para o próximo operando a ser recalculado.
c) Instruções e dados estão em uma memória física única; um programa é constituído de um conjunto de instruções de máquina; uma instrução é lida da memória quando todos os seus operandos-fonte estiverem prontos e disponíveis; essa instrução é executada e o resultado é escrito no operando de destino, passando-se, então, para a instrução seguinte que tiver todos seus operandos disponíveis.
d) Instruções e dados estão em memórias físicas distintas; um programa é constituído de uma sequência de instruções de máquina; uma instrução é lida da memória de acordo com a ordem dessa sequência e, quando é executada, passa-se, então, para a próxima instrução na sequência.

1.4 Considere a máquina RISC-V (32 bits) e a seguinte instrução em Assembly: *add s0, s1, s2*. Qual o código de máquina desta instrução (em hexadecimal)?

a) 0x01248833 b) 0x01428433 c) 0x01248430 d) 0x01248433 e) 0x01248423

1.5 Considere a máquina RISC-V (32 bits) e uma instrução com o seguinte código de máquina (em hexadecimal): 0x01c3c333. Este código corresponde à qual instrução?

a) *addi s0, s1, t3* b) *xor t1, t2, t3* c) *add t0, t3, t4* d) *or t1, t2, t3* e) *sw s2, 4(t1)*

1.6 Analise o trecho de código em linguagem C a seguir.

g = h + a[8];

Em uma linguagem RISC, qual é o código de montagem correspondente?

a) lw t0,32(s1)
add s1,s2,t0

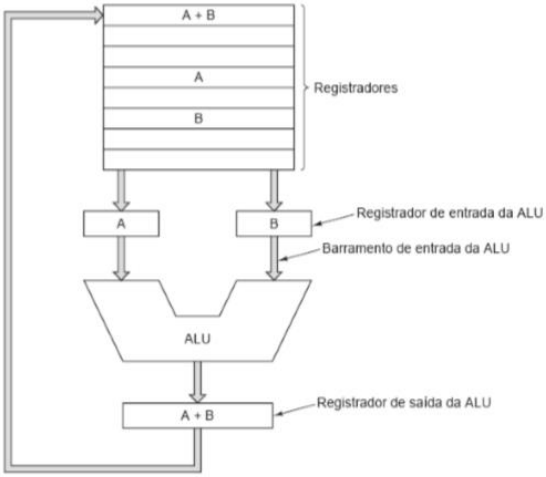
b) lw t0,32(s3)
add s1,s2,s0

c) lw t0,32(s3)
add s1,s3,t0

d) lw t0,32(s2)
add s1,s2,t0

e) lw t0,32(s3)
add s1,s2,t0

1.7 Analise a operação de uma máquina cujo caminho de dados está representado na figura abaixo.



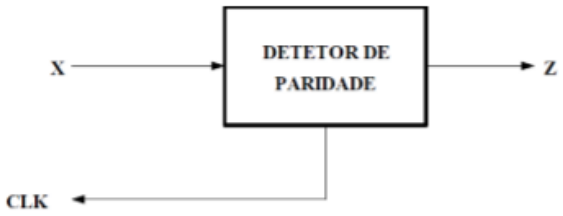
Considere que o tempo de carga dos registradores de entrada da ALU (*Arithmetic and Logic Unit*) é 5 ns, o tempo de execução da instrução na ALU é 10 ns e o armazenamento do resultado de volta no registrador demora 5 ns. Com base nessas informações, assinale a alternativa que contém o número máximo de MIPS que essa máquina é capaz de executar sem o emprego de paralelismo (uma única instrução executada por vez).

a) 20 MIPS.
b) 50 MIPS.
c) 200 MIPS.
d) 10 MIPS.
e) 500 MIPS.

Obs.: MIPS (Milhões de Instruções Por Segundo)

2. Obter a máquina de estados de um circuito que detecta a paridade de um sinal serial. Considere paridade par.

1,0



3. Considere o trecho de código abaixo (Assembly de uma máquina RISC-V, 32 bits) e responda as questões a seguir.

1,5

a) Para cada instrução executada, complete a tabela informando o endereço de memória onde essa será armazenada (incluindo o valor do `target`), a operação realizada e o conteúdo dos registradores envolvidos após sua execução.
b) Substitua `sll s1, s1, s0` por `s11 s1, s1, s1` e informe o resultado final dos registradores envolvidos.
c) Apresente o código de máquina da instrução `add s1, s1, s0`.

Endereço de memória	Instrução	Operação realizada	Conteúdo dos registradores envolvidos
0x00000200	`addi s0, zero, 2`		
	`addi s1, zero, 1`		
	`sll s1, s1, s0`		
	`bne s0, s1, target`		
	`addi s1, s1, 1`		
	`sub s1, s1, s0`		
target:	`add s1, s1, s0`		

4. Considere o trecho de código abaixo (Assembly de uma máquina RISC-V, 32 bits), informe qual a operação realizada e qual o conteúdo dos registradores *s0* e *s1* ao final da execução.

1,0

```
# s0 = i, s1 = sum
addi s1, zero, 0x0
addi s0, zero, 0x0
addi t0, zero, 0x5
for:
 beq s0, t0, done
 add s1, s1, s0
 addi s0, s0, 0x1
j for
done:
```

5. Considere a figura abaixo (tela de um simulador RISC-V para um trecho de código) e responda as questões a seguir.

1,5

a) Descreva a composição do código de máquina da instrução "*li s1, 0*".

b) Descreva a composição do código de máquina da instrução "*beq s0, t0, pc + 16*".

c) Informe o valor do Registrador pc (Contador de Programa) ao ser executada a instrução "***j pc -0xc***" e indique qual é a próxima instrução a ser executada após esta.

```
0x00000200 (0x00000493) li      s1, 0
0x00000204 (0x00000413) li      s0, 0
0x00000208 (0x00500293) li      t0, 5
0x0000020c (0x00540863) beq     s0, t0, pc + 16
0x00000210 (0x008484b3) add     s1, s1, s0
0x00000214 (0x00140413) addi    s0, s0, 1
0x00000218 (0xff5ff06f) j       pc - 0xc
```

6. Para o trecho de código a seguir, em Assembly RISC-V (32 bits), informe qual é o código correspondente em linguagem de alto nível.

1,5

```
# s0 = f, s1 = g, s2 = h, s3 = i, s4 = j
  bne s3, s4, else
  add s0, s1, s2
 j  L2
else:
 sub s0, s0, s3
L2:
```

SUCESSO !

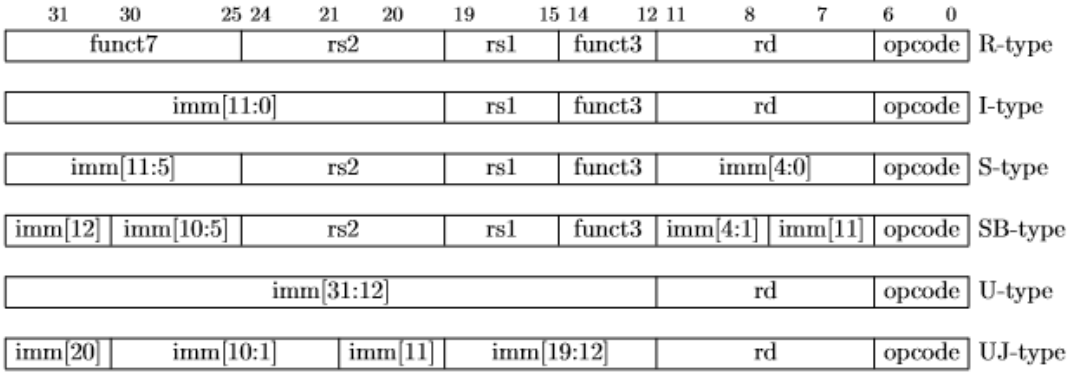
Arquitetura RISC-V
RV32I: Exemplos de Instruções

Instruction	Format	Meaning
`add rd, rs1, rs2`	R	Add registers
`sub rd, rs1, rs2`	R	Subtract registers
`sll rd, rs1, rs2`	R	Shift left logical by register
`srl rd, rs1, rs2`	R	Shift right logical by register
`sra rd, rs1, rs2`	R	Shift right arithmetic by register
`and rd, rs1, rs2`	R	Bitwise AND with register
`or rd, rs1, rs2`	R	Bitwise OR with register
`xor rd, rs1, rs2`	R	Bitwise XOR with register
`slt rd, rs1, rs2`	R	Set if less than register, 2's complement
`sltu rd, rs1, rs2`	R	Set if less than register, unsigned
`addi rd, rs1, imm[11:0]`	I	Add immediate
`slli rd, rs1, shamt[4:0]`	I	Shift left logical by immediate
`srli rd, rs1, shamt[4:0]`	I	Shift right logical by immediate
`srai rd, rs1, shamt[4:0]`	I	Shift right arithmetic by immediate
`andi rd, rs1, imm[11:0]`	I	Bitwise AND with immediate
`ori rd, rs1, imm[11:0]`	I	Bitwise OR with immediate
`xori rd, rs1, imm[11:0]`	I	Bitwise XOR with immediate
`slti rd, rs1, imm[11:0]`	I	Set if less than immediate, 2's complement
`sltiu rd, rs1, imm[11:0]`	I	Set if less than immediate, unsigned
`lui rd, imm[31:12]`	U	Load upper immediate
`auipc rd, imm[31:12]`	U	Add upper immediate to `pc`

Instruction	Format	Meaning
`lb rd, imm[11:0](rs1)`	I	Load byte, signed
`lbu rd, imm[11:0](rs1)`	I	Load byte, unsigned
`lh rd, imm[11:0](rs1)`	I	Load half-word, signed
`lhu rd, imm[11:0](rs1)`	I	Load half-word, unsigned
`lw rd, imm[11:0](rs1)`	I	Load word
`sb rs2, imm[11:0](rs1)`	S	Store byte
`sh rs2, imm[11:0](rs1)`	S	Store half-word
`sw rs2, imm[11:0](rs1)`	S	Store word
`fence pred, succ`	I	Memory ordering fence
`fence.i`	I	Instruction memory ordering fence

Instruction	Format	Meaning
`beq rs1, rs2, imm[12:1]`	SB	Branch if equal
`bne rs1, rs2, imm[12:1]`	SB	Branch if not equal
`blt rs1, rs2, imm[12:1]`	SB	Branch if less than, 2's complement
`bltu rs1, rs2, imm[12:1]`	SB	Branch if less than, unsigned
`bge rs1, rs2, imm[12:1]`	SB	Branch if greater or equal, 2's complement
`bgeu rs1, rs2, imm[12:1]`	SB	Branch if greater or equal, unsigned
`jal rd, imm[20:1]`	UJ	Jump and link
`jalr rd, rs1, imm[11:0]`	I	Jump and link register

Arquitetura RISC-V
RV32I: Formatos das Instruções



Arquitetura RISC-V
RV32I: Registradores

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5–7	t0–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller

Arquitetura RISC-V
Pseudoinstruções

Pseudoinstruction	Base Instruction(s)	Meaning
`la rd, symbol`	`auipc rd, symbol[31:12]` `addi rd, rd, symbol[11:0]`	Load address
`l{b|h|w|d} rd, symbol`	`auipc rd, symbol[31:12]` `l{b|h|w|d} rd, symbol[11:0](rd)`	Load global
`s{b|h|w|d} rd, symbol, rt`	`auipc rt, symbol[31:12]` `s{b|h|w|d} rd, symbol[11:0](rt)`	Store global
`fl{w|d} rd, symbol, rt`	`auipc rt, symbol[31:12]` `fl{w|d} rd, symbol[11:0](rt)`	Floating-point load global
`fs{w|d} rd, symbol, rt`	`auipc rt, symbol[31:12]` `fs{w|d} rd, symbol[11:0](rt)`	Floating-point store global
`nop`	`addi x0, x0, 0`	No operation
`li rd, immediate`	*Myriad sequences*	Load immediate
`mv rd, rs`	`addi rd, rs, 0`	Copy register
`not rd, rs`	`xori rd, rs, -1`	One's complement
`neg rd, rs`	`sub rd, x0, rs`	Two's complement
`negw rd, rs`	`subw rd, x0, rs`	Two's complement word
`sext.w rd, rs`	`addiw rd, rs, x0`	Sign extend word
`seqz rd, rs`	`sltiu rd, rs, 1`	Set if = zero
`snez rd, rs`	`sltu rd, x0, rs`	Set if ≠ zero
`sltz rd, rs`	`slt rd, rs, x0`	Set if < zero
`sgtz rd, rs`	`slt rd, x0, rs`	Set if > zero
`fmv.s rd, rs`	`fsgnj.s rd, rs, rs`	Copy single-precision register
`fabs.s rd, rs`	`fsgnjx.s rd, rs, rs`	Single-precision absolute value
`fneg.s rd, rs`	`fsgnjn.s rd, rs, rs`	Single-precision negate
`fmv.d rd, rs`	`fsgnj.d rd, rs, rs`	Copy double-precision register
`fabs.d rd, rs`	`fsgnjx.d rd, rs, rs`	Double-precision absolute value
`fneg.d rd, rs`	`fsgnjn.d rd, rs, rs`	Double-precision negate
`beqz rs, offset`	`beq rs, x0, offset`	Branch if ≠ zero
`bnez rs, offset`	`bne rs, x0, offset`	Branch if = zero
`blez rs, offset`	`bge x0, rs, offset`	Branch if ≤ zero
`bgez rs, offset`	`bge rs, x0, offset`	Branch if ≥ zero
`bltz rs, offset`	`blt rs, x0, offset`	Branch if < zero
`bgtz rs, offset`	`blt x0, rs, offset`	Branch if > zero
`j offset`	`jal x0, offset`	Jump
`jal offset`	`jal x1, offset`	Jump and link
`jr rs`	`jalr x0, rs, 0`	Jump register
`jalr rs`	`jalr x1, rs, 0`	Jump and link register
`ret`	`jalr x0, x1, 0`	Return from subroutine
`call offset`	`auipc x6, offset[31:12]` `jalr x1, x6, offset[11:0]`	Call far-away subroutine
`tail offset`	`auipc x6, offset[31:12]` `jalr x0, x6, offset[11:0]`	Tail call far-away subroutine

Arquitetura RISC-V

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
0000 | pred	succ	00000	000	00000	0001111	FENCE
0000 | 0000	0000	00000	001	00000	0001111	FENCE.I
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK