PUCRS ESCOLA POLITÉCNICA

4646B-04

FUNDAMENTOS DE SISTEMAS DIGITAIS

UNIDADE 4 – Projeto de Final de Disciplina

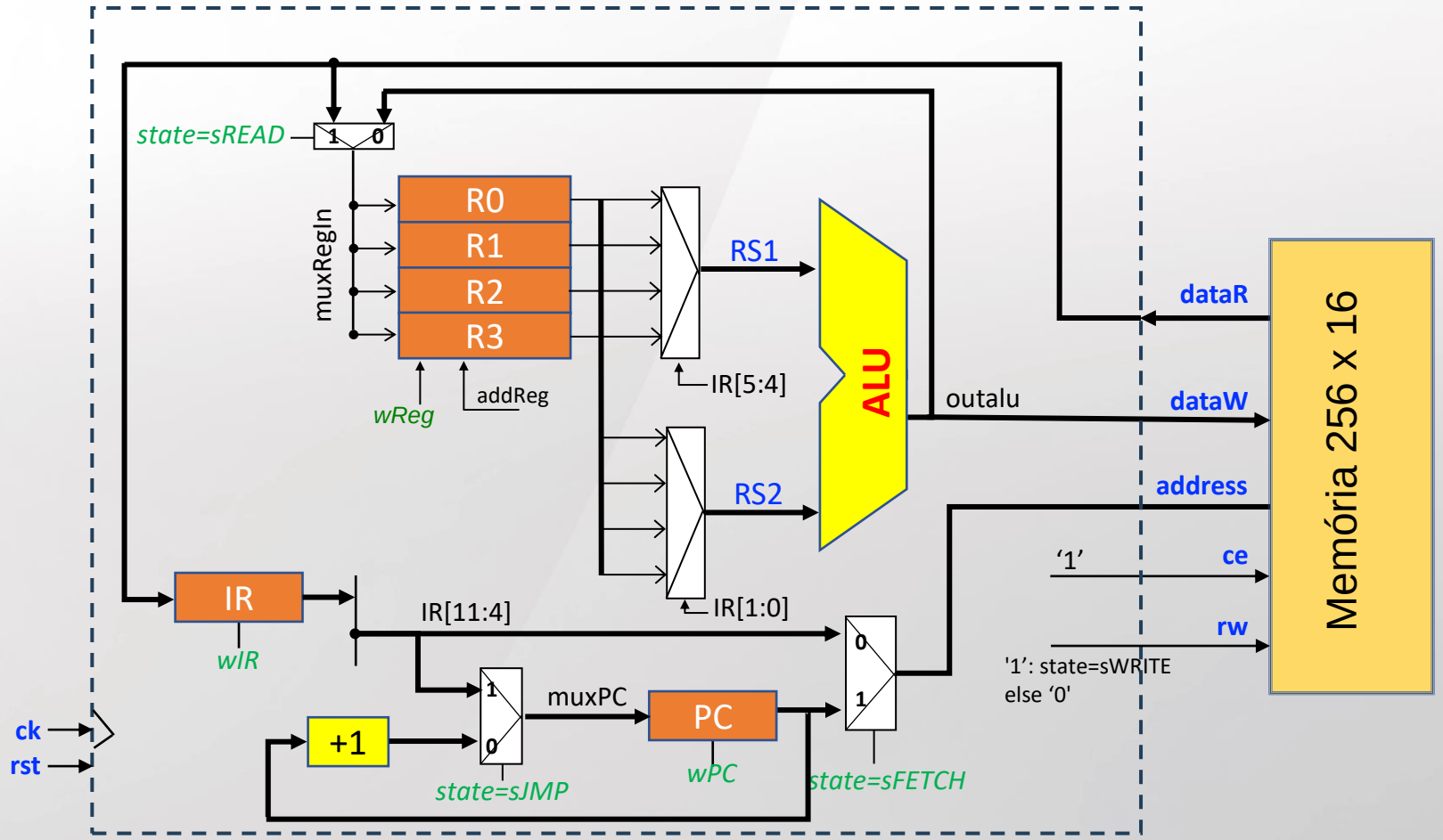
Prof. Me. Marlon Moraes

MARISTA

Nano CPU

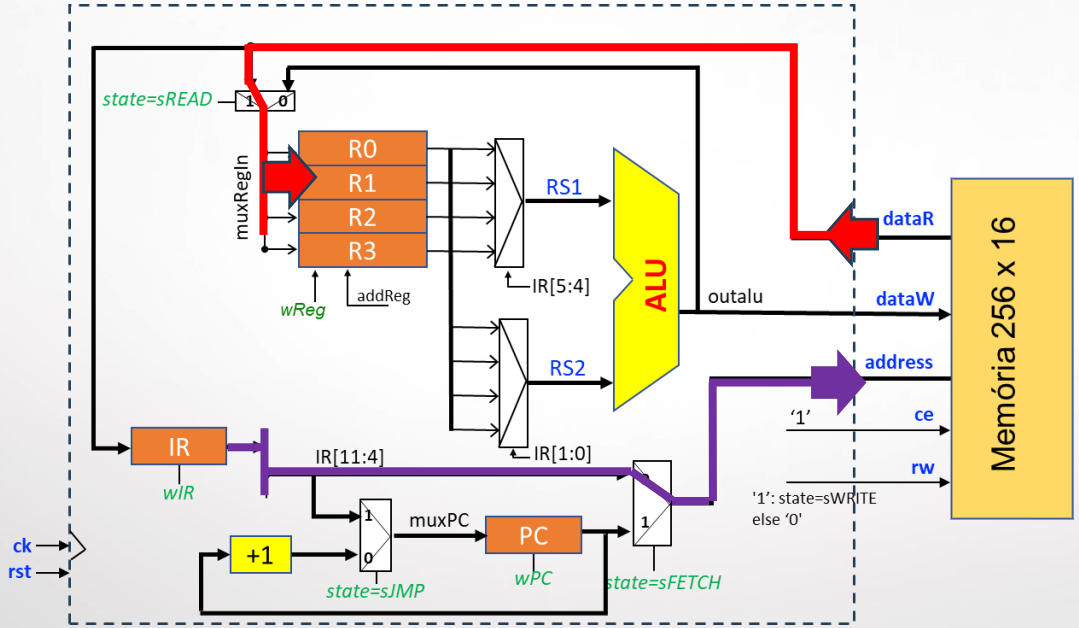
Exemplo de processador muito simples, porém seguindo conceitos importantes em arquitetura de processadores:

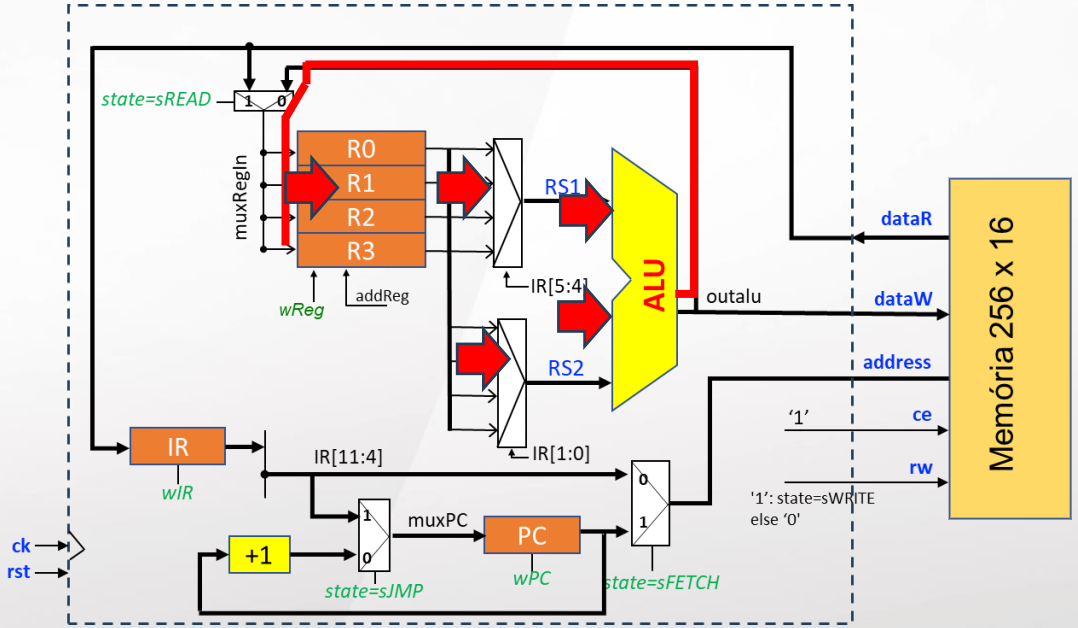
1. Operações lógicas e aritméticas executadas entre registradores
   Arquitetura ***load-store***
2. Instruções de formato fixo



- **R0 a R3 → registradores de propósito geral**
- **PC: *program counter* – endereço atual do programa na memória**
- **IR: *instruction register* – instrução atual**

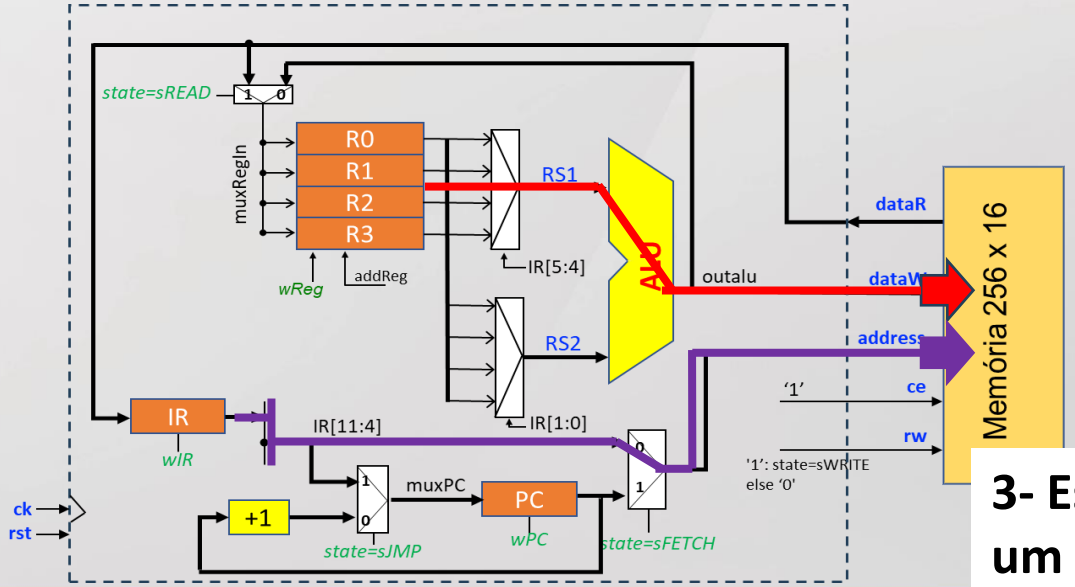
Como processar os dados armazenados em memória?





1- Le operandos da memória

- IR contém instrução atual, e nesta instrução há o endereço da memória
- Dado lido é escrito em um registrador



2 - Operações lógico-aritméticas

R*x* ← RS1 *op* RS2

3- Escreve o conteúdo de um registrador na memória

Nano CPU – conjunto de instruções

- O registrador IR contém **instruções** de formato **fixo**:
  - Bits 15:12 – 4 bits que especificam a instrução
  - Bits 11:4 – podem indicar um endereço de 8 bits ou endereços de registradores
  - Bits 3:0 – endereço de registrador

Instrução	15:12	11:8	7:4	3:0	Operação
iREAD	0	endereço		RS2	RS2 ← PMEM(end)
iWRITE	1	endereço		RS2	PMEM(end) ← RS2
iJMP	2	endereço		0	PC ← end
iBRANCH	3	endereço		RS2	PC ← end **se** RS2=1
iXOR	4	**Rt**	RS1	RS2	Rt ← RS1 xor RS2
iSUB	5	**Rt**	RS1	RS2	Rt ← RS1 - RS2
iADD	6	**Rt**	RS1	RS2	Rt ← RS1 + RS2
iLESS	7	**Rt**	RS1	RS2	Rt ← 1 **se** RS1 < RS2 senão 0
...	...				*a ser incluído pelos alunos*
iEND	15 $(F)_{16}$	0	0	0	termina a execução

Notar que podemos ter até 16 registradores de propósito geral – por que?

Programando o processador

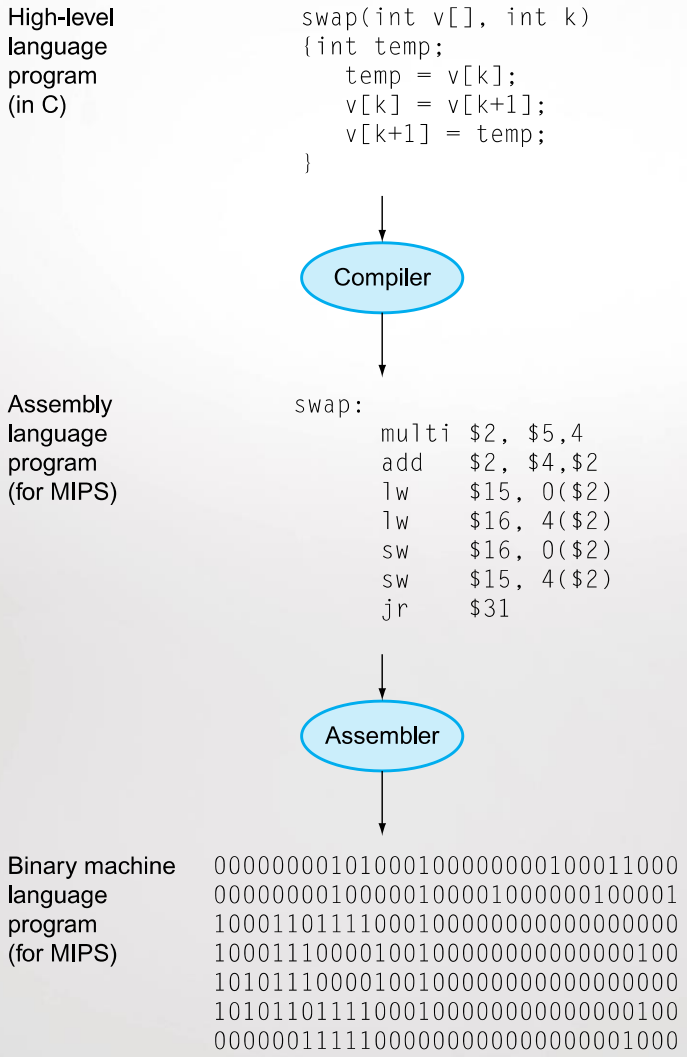


FIGURE 1.4 C program compiled into assembly language and then assembled into binary machine language. Although the translation from high-level language to binary machine language is shown in two steps, some compilers cut out the middleman and produce binary machine language directly. These languages and this program are examined in more detail in Chapter 2.

- A memória contém o programa a ser executado, assim como os dados utilizados no processamento
- Programar direto em binário é impraticável – por isto os processadores usam um linguagem chamada ***assembly,*** única para cada processador

Linguagem ***assembly***: descrição de um programa utilizando instruções suportadas pela arquitetura (**ISA** – *Instruction Set Architecture)*

Código binário que vai para a memória do processador

Exemplo de programa (sintaxe informal)

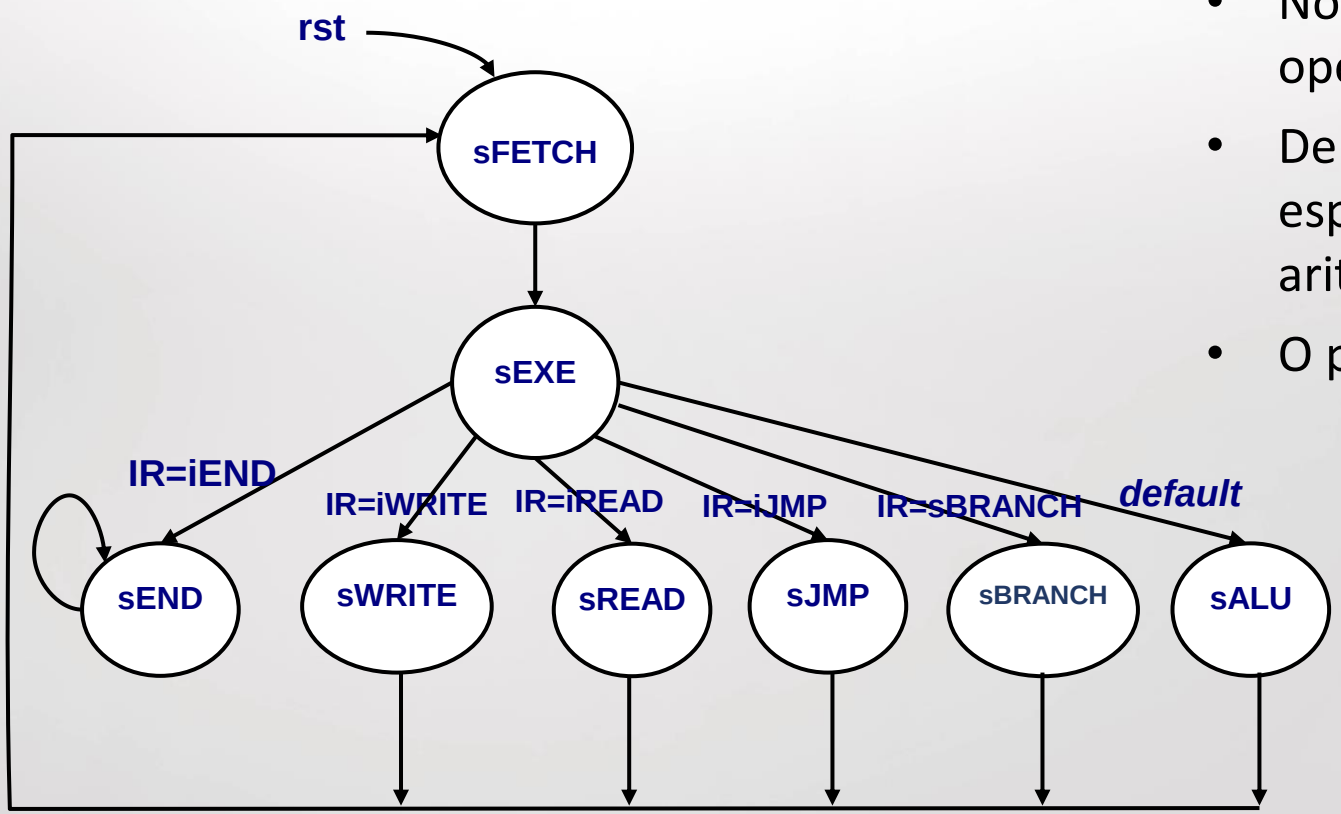
```
i = 0
a =0
do {
        a = a + i
        i = i + 1
} while (i < 10)
"write" a
fim
```

Instrução	15:12	11:8	7:4	3:0
iREAD	0	endereço		RS2
iWRITE	1	endereço		RS2
iJMP	2	endereço		0
iBRANCH	3	endereço		RS2
iXOR	4	Rt	RS1	RS2
iSUB	5	Rt	RS1	RS2
iADD	6	Rt	RS1	RS2
iLESS	7	Rt	RS1	RS2
...	...			
iEND	15 $(F)_{16}$	0	0	0

Endereço	Instrução - *assembly*	comentário	Binário (hexa)
0	**read** R0, 10	Le da posição 10: R0←0 (i)	0 0A 0
1	**read** R1, 10	Le da posição 10: R1←0 (a)	0 0A 1
2	**read** R3, 12	Le da posição 12: R3 ← 10	0 0C 3
3	**add** R1, R1, R0	a = a + i	6 1 1 0
4	**read** R2, 11	Le da posição 11: R2 ← 1	0 0B 2
5	**add** R0, R0, R2	i = i + 1	6 0 0 2
6	**less** R2, R0, R3	R2 ← 1 se i < 10 else 0	7 2 0 3
7	**branch** 3, R2	se R2 = 1 salta para 3	3 0 3 2
8	**write** R1, 13	Escreve a na posição 13	1 0D 1
9	**end**	Termina a execução	F000
10 $(A)_{16}$	0		0000
11 $(B)_{16}$	1		0001
12 $(C)_{16}$	10		000A
13 $(D)_{16}$		Receberá o valor de a	0000

Como o processador executa um programa?

Máquina de estados para sequenciamento das instruções e sinais de controle

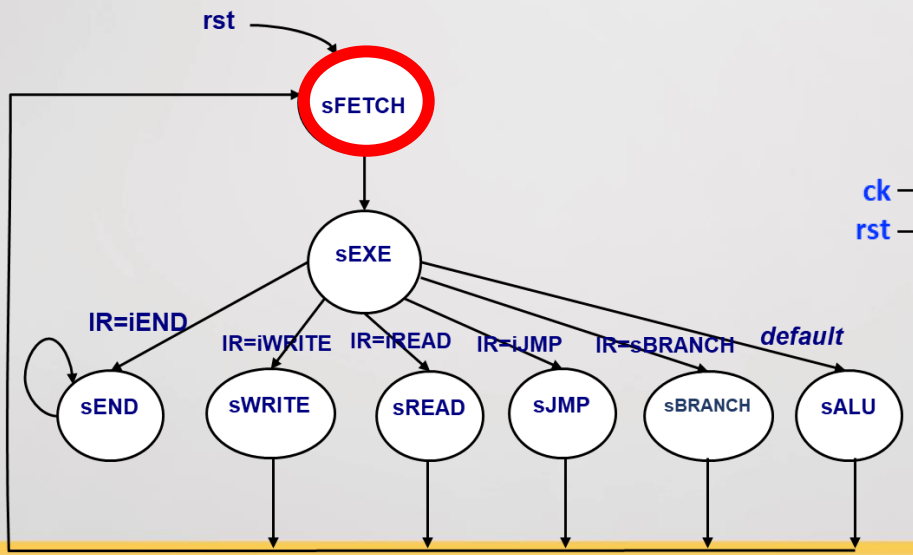


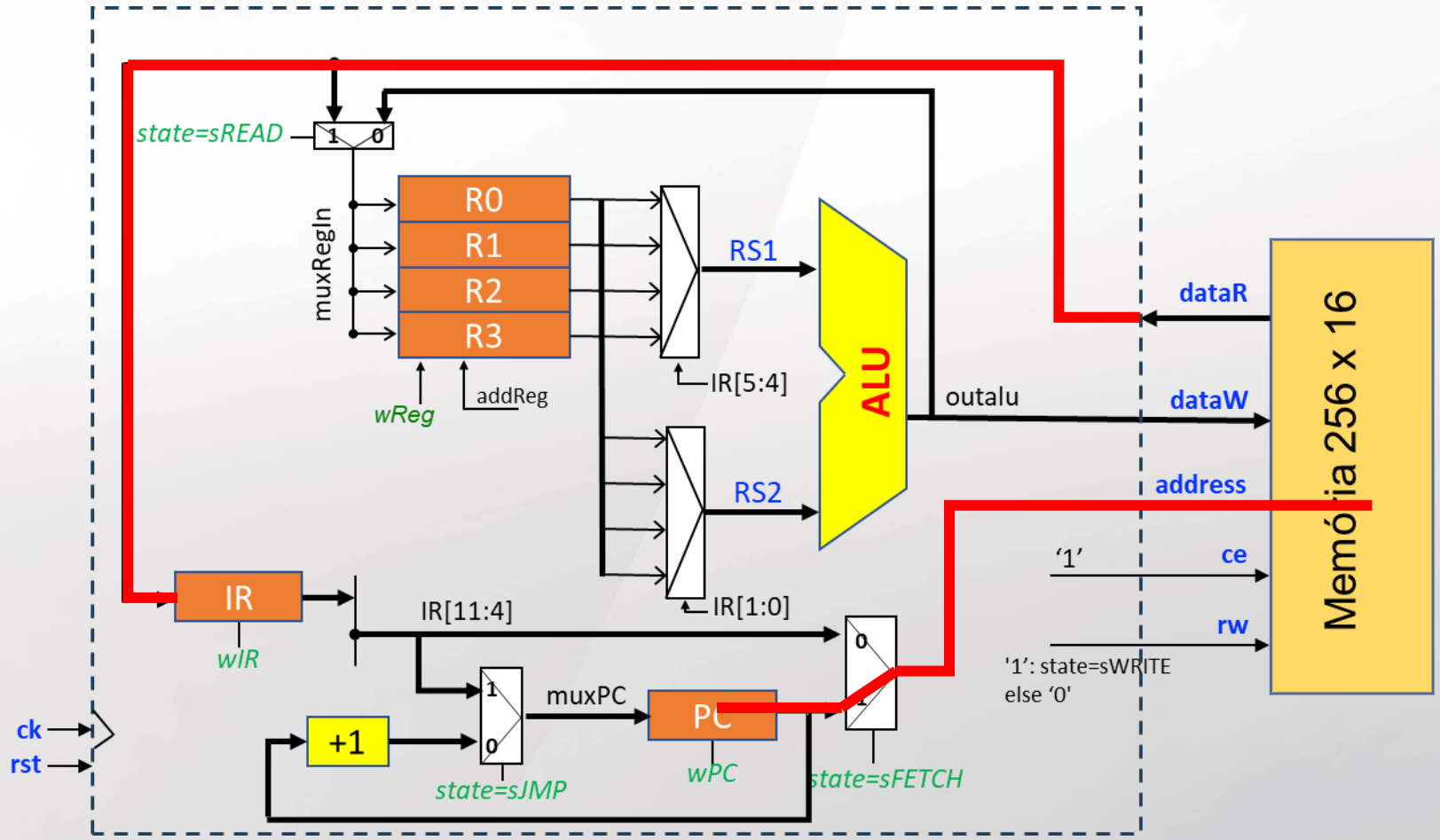
- Todas as instruções executam o estado de ***sFETCH*** para ler da memória a instrução apontada pelo **PC** e gravar este dado no **IR**
- No estado ***sEXEC*** é feita a leitura da memória ou a operação na ALU
- Depois cada instrução é concluída em um estado específico (por default as instrução lógicas e aritméticas são executadas no estado ***sALU***)
- O programa termina ao encontrar a instrução **iEND**

Busca da instrução – igual para todas as instruções

Neste estado de ***sFETCH***:

- PC endereça a memória
- Dado é lido da memória
- Ativa o *wIR* para escrita em IR



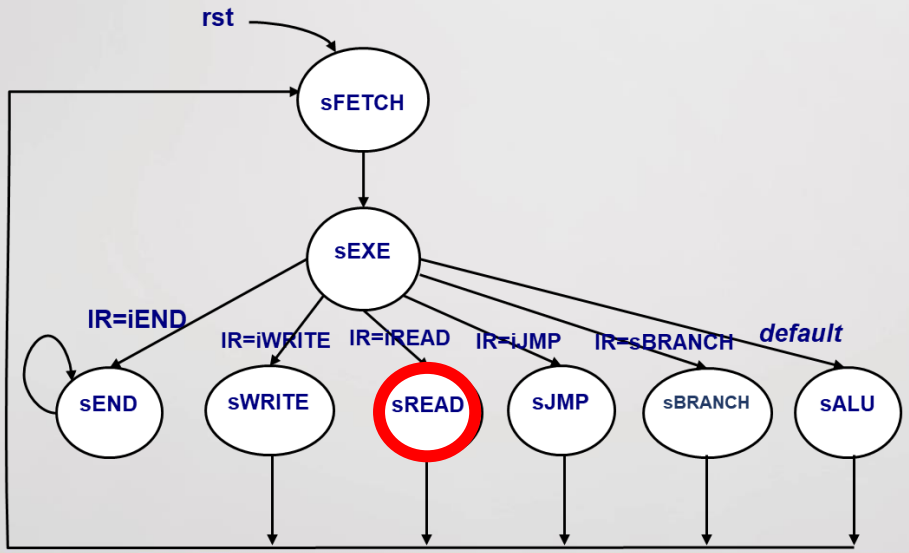


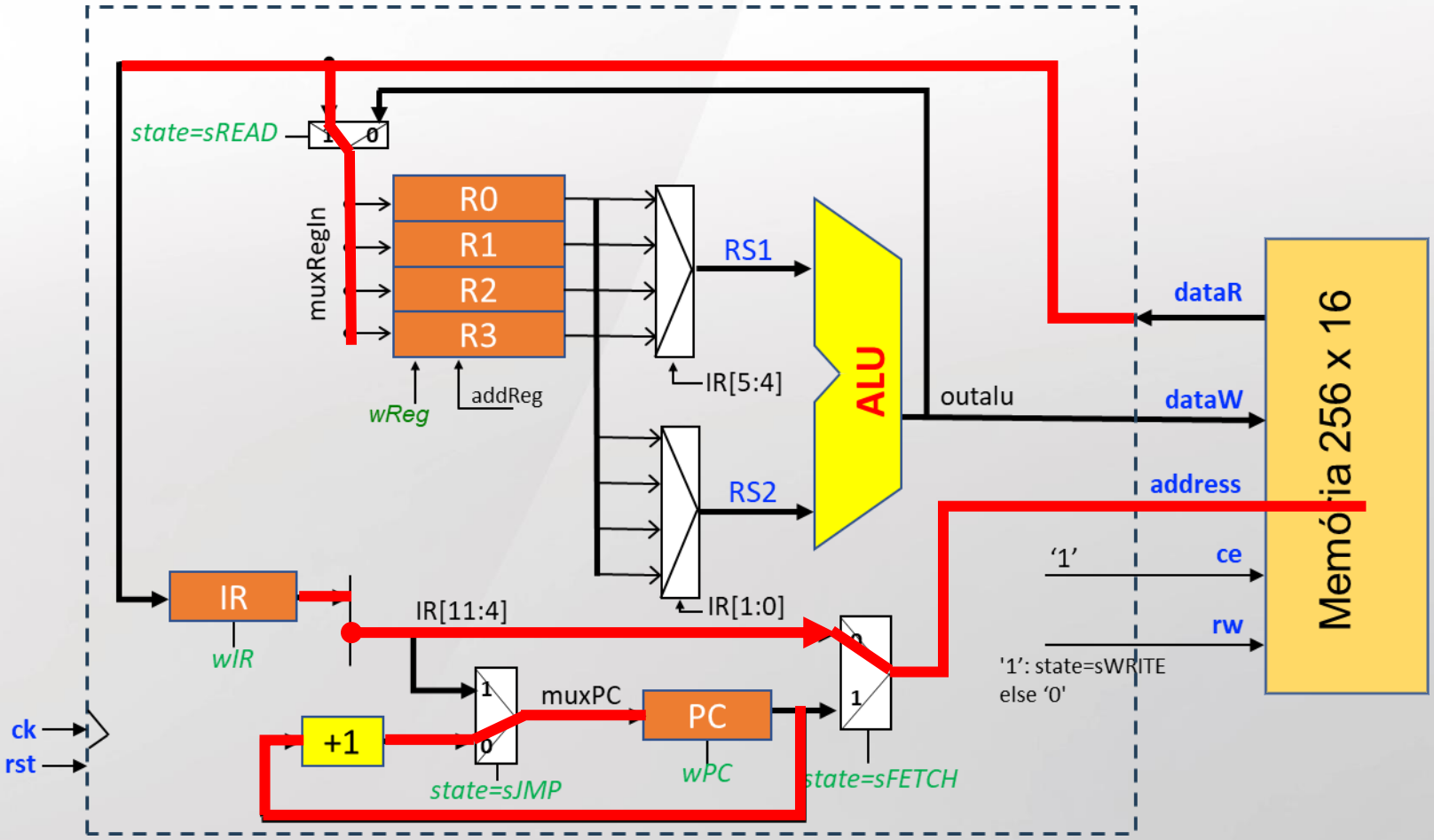
Busca da instrução - *sFETCH*

Execução do sREAD

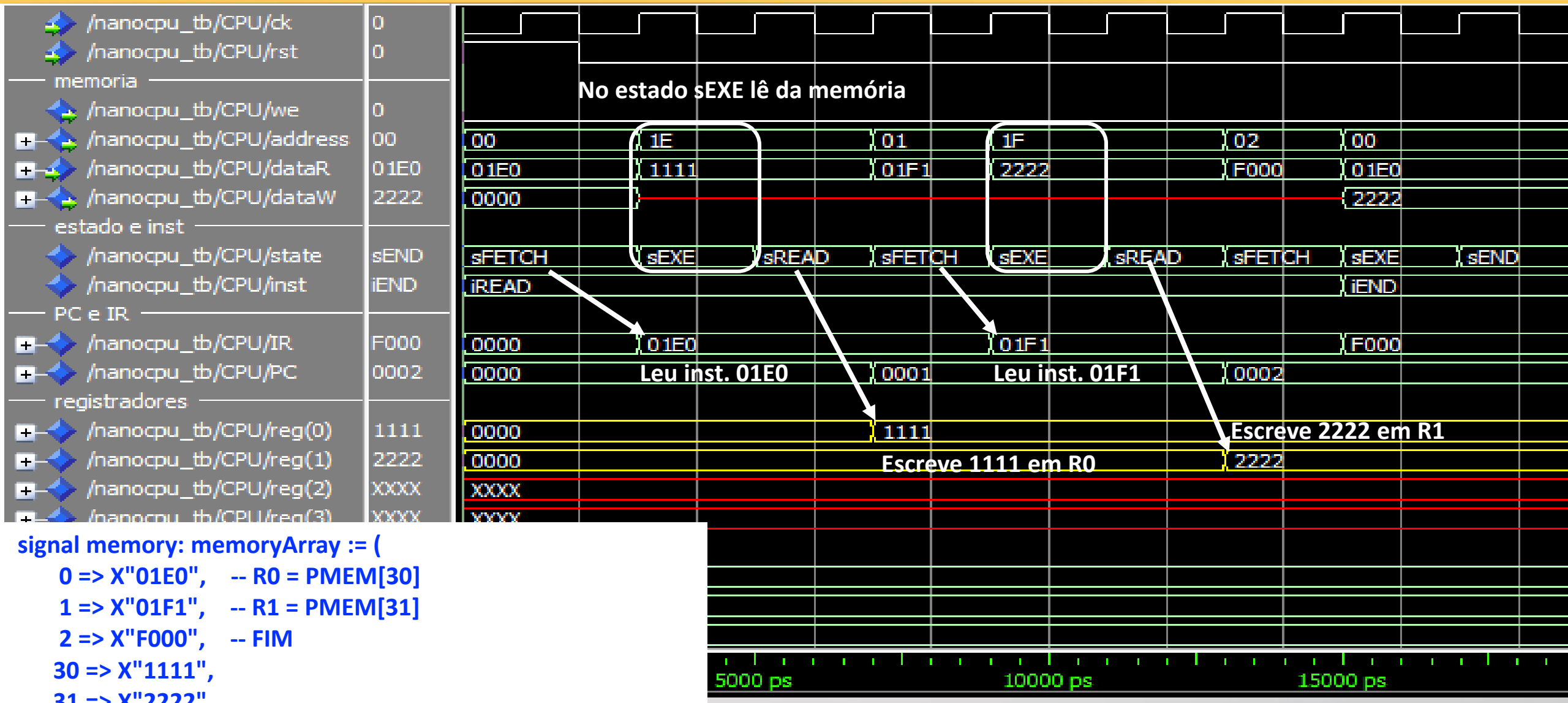
No estado ***sREAD***:

- IR[11:4] endereça a memória
- Dado é lido da memória
- Escreve no banco de registradores ativando *wReg* (registrador alterado especificado em IR[1:0])
- Incrementa o PC





Inicializando dois registradores

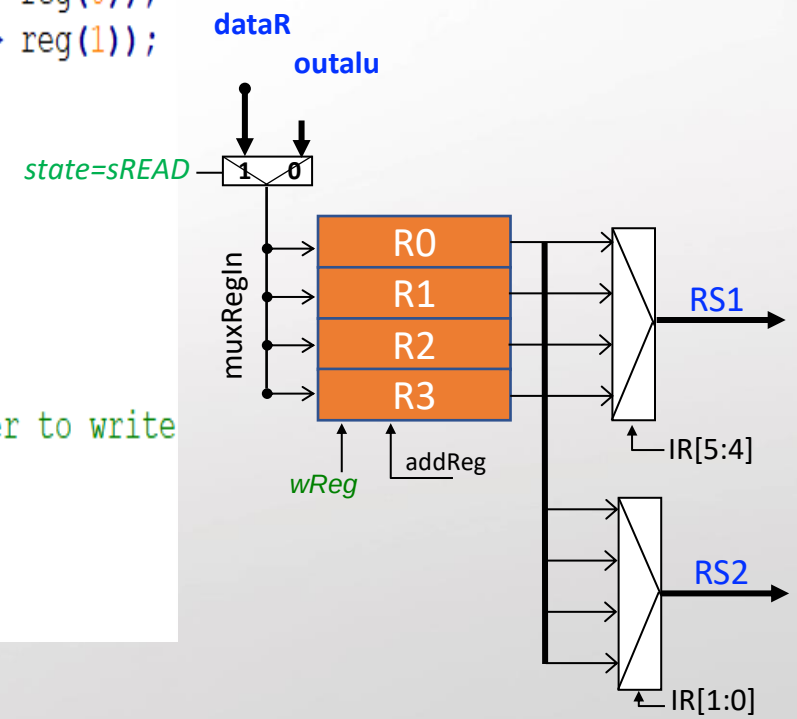



```
signal memory: memoryArray := (
     0 => X"01E0",    -- R0 = PMEM[30]
     1 => X"01F1",    -- R1 = PMEM[31]
     2 => X"F000",    -- FIM
    30 => X"1111",
    31 => X"2222",
    others => (others => '0')
  );
```

O *signal* memoryArray armazena o binário (programa)
a estrutura é <endereço> => <valor>

Atividade 1 Completar o banco de registradores (a)

```
83    -- register bank - 4 general purpose registers
84    --
85    r0 : entity work.Reg16bit port map(ck => ck, rst => rst, we => wen(0), D => muxRegIn, Q => reg(0));
86    r1 : entity work.Reg16bit port map(ck => ck, rst => rst, we => wen(1), D => muxRegIn, Q => reg(1));
87   --complete
88                    Habilitação de escrita para cada registrador
89    wen(0) <= '1' when addReg = "00" and wReg = '1' else '0';
90    wen(1) <= '1' when addReg = "01" and wReg = '1' else '0';
91   --complete
92   --complete
93
94    addReg <= IR(1 downto 0) when state = sREAD else IR(9 downto 8);   -- index of the register to write
95    muxRegIn <= dataR when state = sREAD else outalu;
96
97    RS1 <= reg(CONV_INTEGER(IR(5 downto 4)));       -- multiplexers to read registers
98    RS2 <= reg(CONV_INTEGER(IR(1 downto 0)));
```



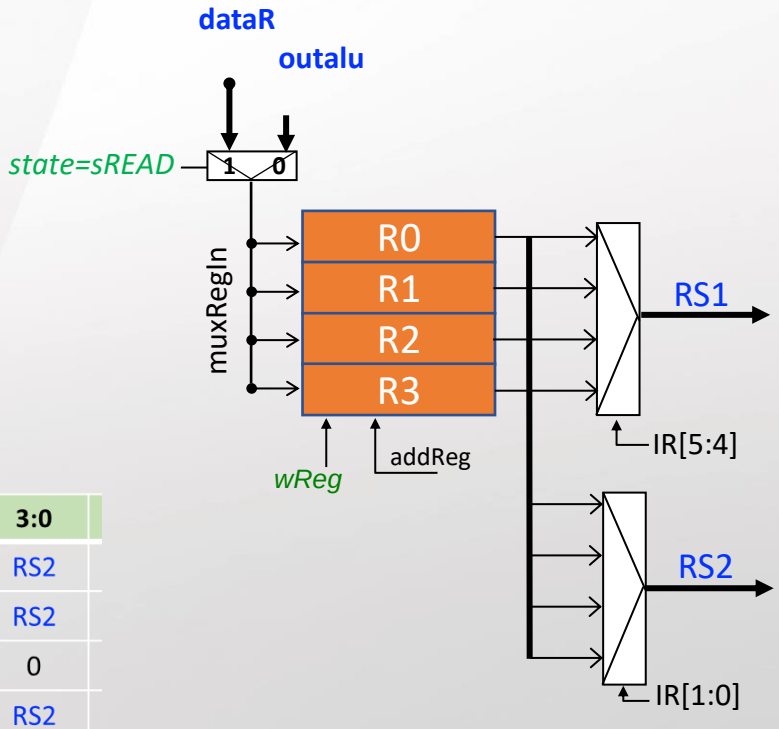
- Linhas 85:86 – registradores r0 e r1
- Linhas 89:90 – habilitação de escrita no registradores
- Linha 94 – mux que define o endereço de qual registrador vai ser alterado
- Linha 95 – mux que define a origem dos dados a serem escritos
- Linhas 97:98 – muxs que definem os registradores que irão para a ALU

Instrução	15:12	11:8	7:4	3:0
iREAD	0	endereço		RS2
iWRITE	1	endereço		RS2
iJMP	2	endereço		0
iBRANCH	3	endereço		RS2
iXOR	4	Rt	RS1	RS2

Atividade 1: completar o banco de registradores (b)

- Alterado o banco de registradores, executar o programa para inicializar os 4 registradores (alterando o *test bench*):

```
signal memory: memoryArray := (
    0 => X"01E0",   -- R0 = PMEM[30]
    1 => X"01F1",   -- R1 = PMEM[31]
    2 => completar,   -- R2 = PMEM[32]
    3 => completar,   -- R3 = PMEM[33]
    4 => X"F000",   -- FIM
    30 => X"1111",
    31 => X"2222",
    32 => X”3333",
    33 => X”4444",
    others => (others => '0')
);
```



Instrução	15:12	11:8	7:4	3:0
iREAD	0	endereço		RS2
iWRITE	1	endereço		RS2
iJMP	2	endereço		0
iBRANCH	3	endereço		RS2
iXOR	4	Rt	RS1	RS2
iSUB	5	Rt	RS1	RS2
iADD	6	Rt	RS1	RS2
iLESS	7	Rt	RS1	RS2
...	...			
iEND	15 $(F)_{16}$	0	0	0

PUCRS ESCOLA POLITÉCNICA

Atividade 1: completar o banco de registradores (c)