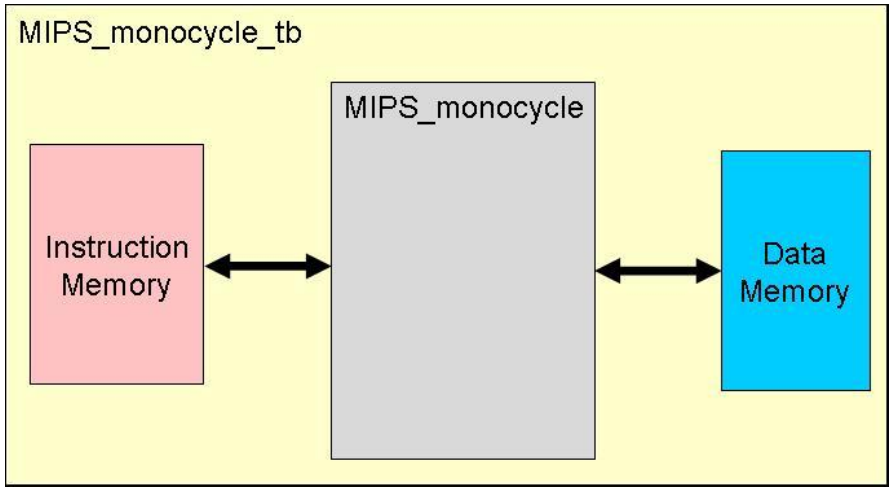
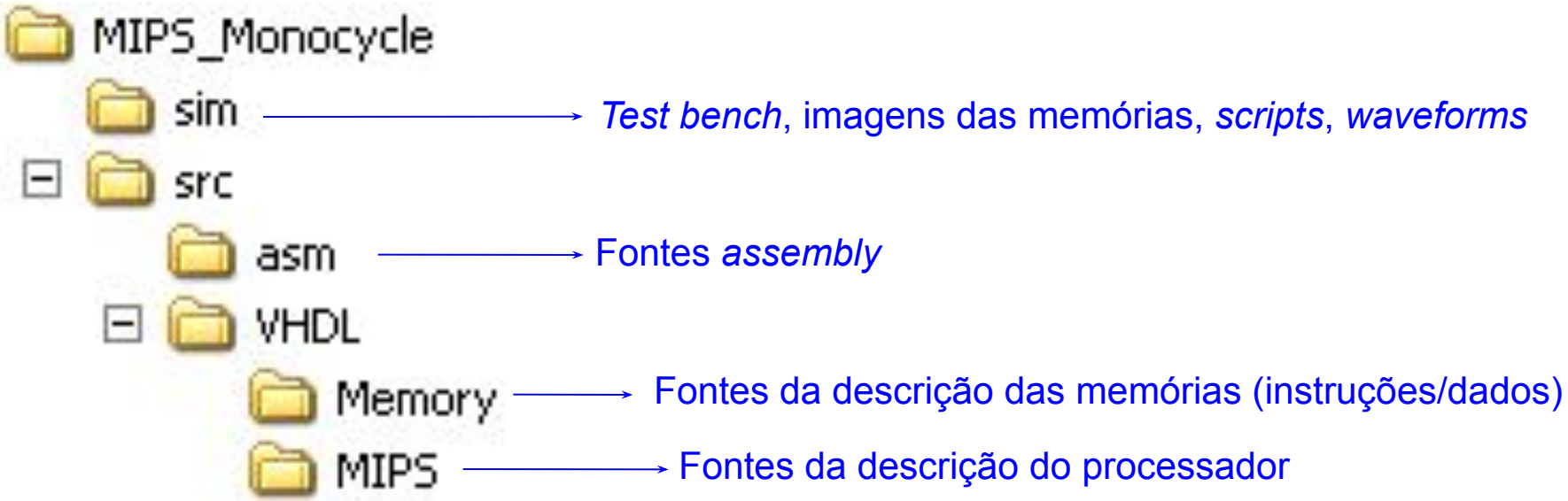
Trabalho 1 - parte 1

- Partiremos de uma organização monociclo do processador MIPS que implementa as seguintes instruções
  - Lógicas/Aritméticas: ADDU, ADDIU, SUBU, AND, OR, ORI
  - Acesso à memória: LW, SW
  - Comparação: SLT
  - Desvio: BEQ, J, JR, JAL
  - Carga de registrador: LUI
- O processador está descrito em VHDL comportamental



Trabalho 1 - parte 1

- Estrutura de diretórios da descrição VHDL

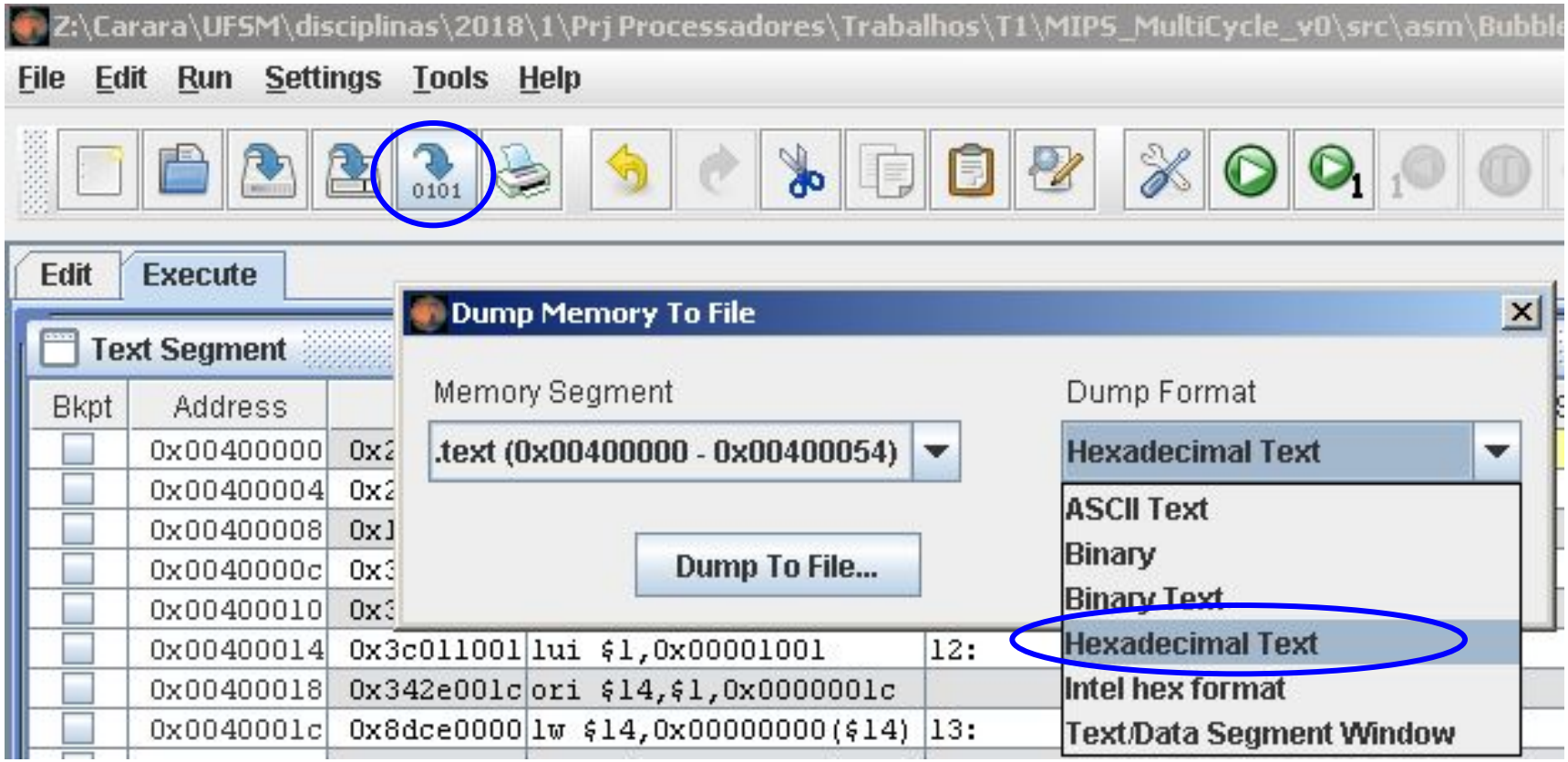


Trabalho 1 - parte 1

- Carga das memórias para simulação VHDL
  - As imagens das memórias (dados e instruções) devem ser geradas a partir do simulador MARS
  - O MARS considera uma memória única para dados e instruções sendo que o programa inicia no endereço 0x00400000 e os dados no endereço 0x10010000
    - O PC é inicializado em 0x0040000
  - Na implementação VHDL o programa é carregado na memória de instruções a partir do endereço 0x00000000 e os dados são carregados na memória de dados a partir do endereço 0x00000000
  - A descrição VHDL da memória faz o mapeamento dos endereços gerados pelo MARS utilizando o parâmetro *generic* OFFSET

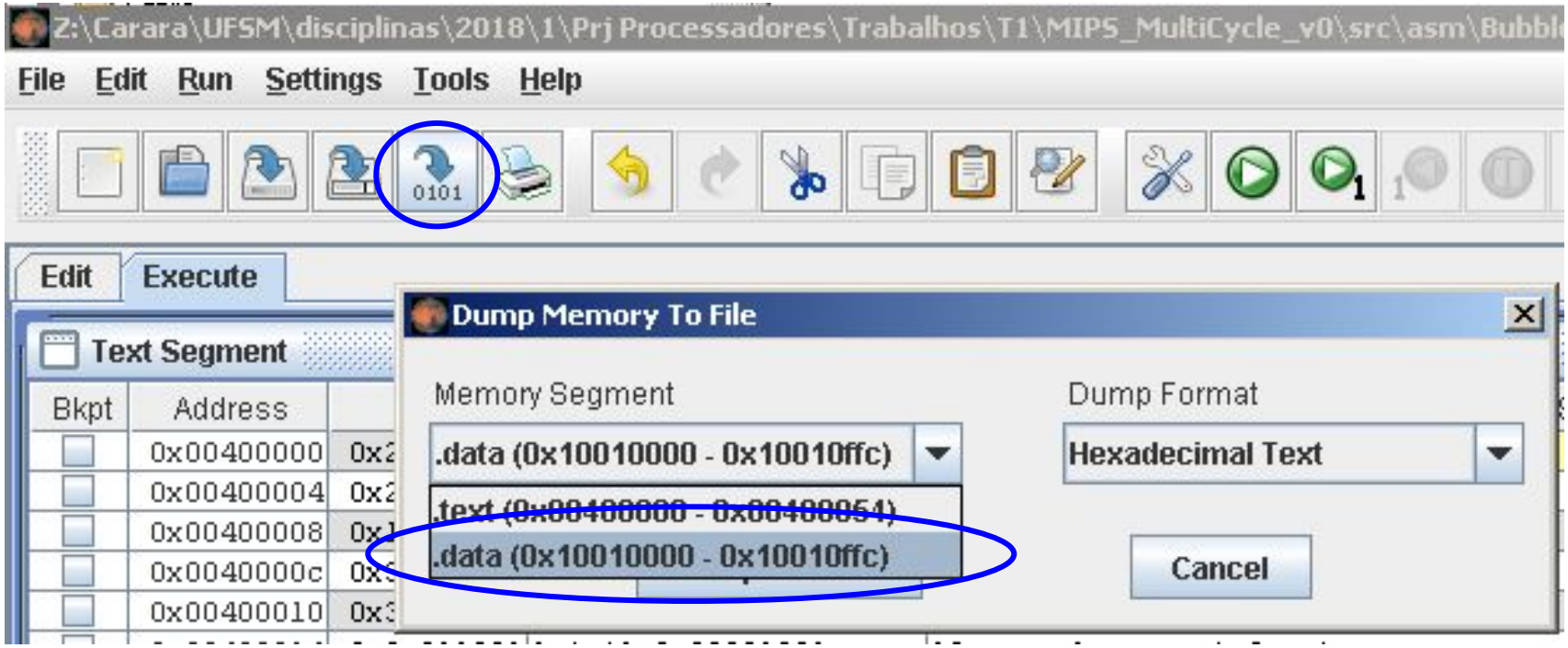
Trabalho 1 - parte 1

- Gerar imagem da memória de instruções
  - MARS: *Dump machine code or data*



Trabalho 1 - parte 1

- Gerar imagem da memória de dados
  - MARS: *Dump machine code or data*



Trabalho 1 - parte 1

- Setar a imagem de cada memória no *test bench* (MIPS_monocycle_tb.vhd)

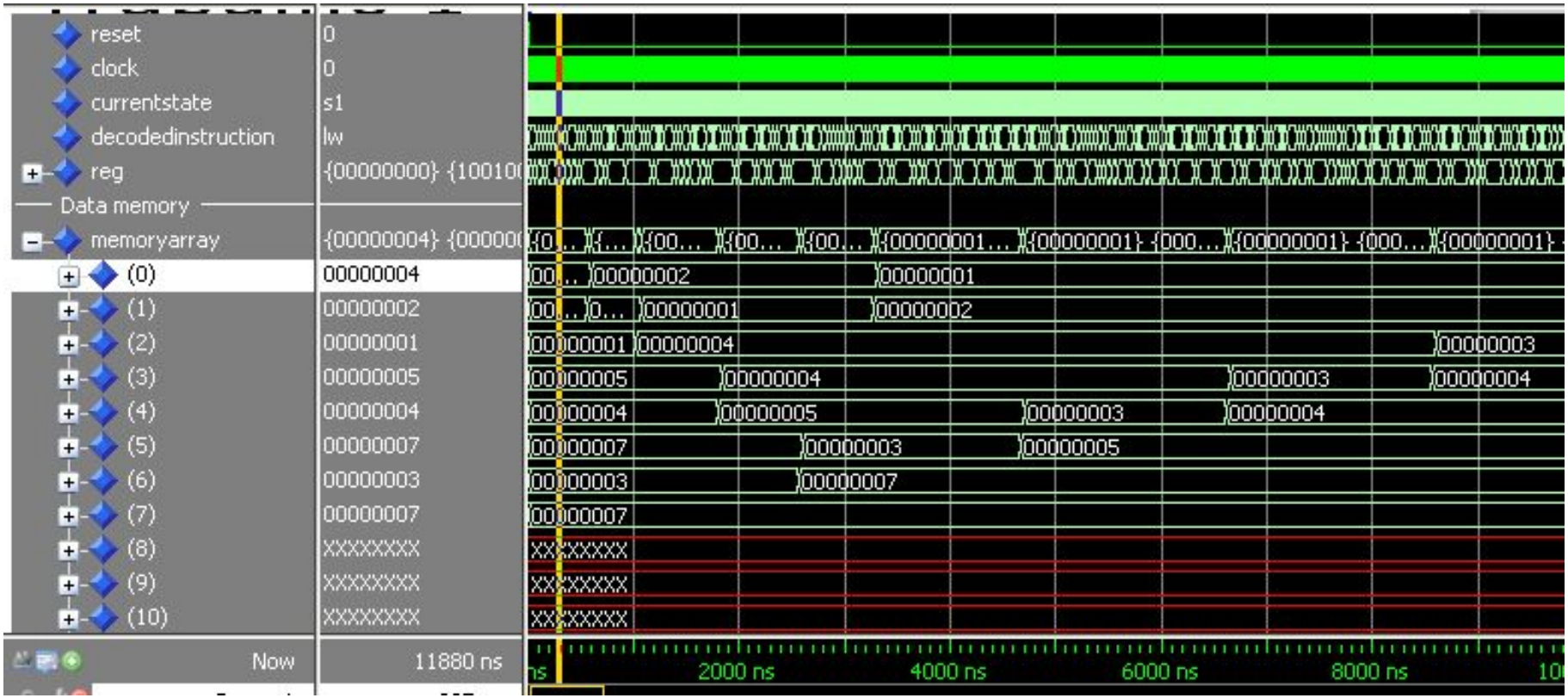
```
INSTRUCTION_MEMORY: entity work.Memory(blockRAM)
    generic map (
        SIZE              => 22,     -- Memory depth
        OFFSET            => PC_START_ADDRESS,
        imageFileName     => "BubbleSort_code.txt"
    )

DATA_MEMORY: entity work.Memory(blockRAM)
    generic map (
        SIZE              => 32,     -- Memory c
        OFFSET            => x"10010000",
        imageFileName     => "BubbleSort_data.txt"
)
```

IMPORTANTE!
A profundidade da memória deve ser menor ou igual ao número de linhas do arquivo de imagem

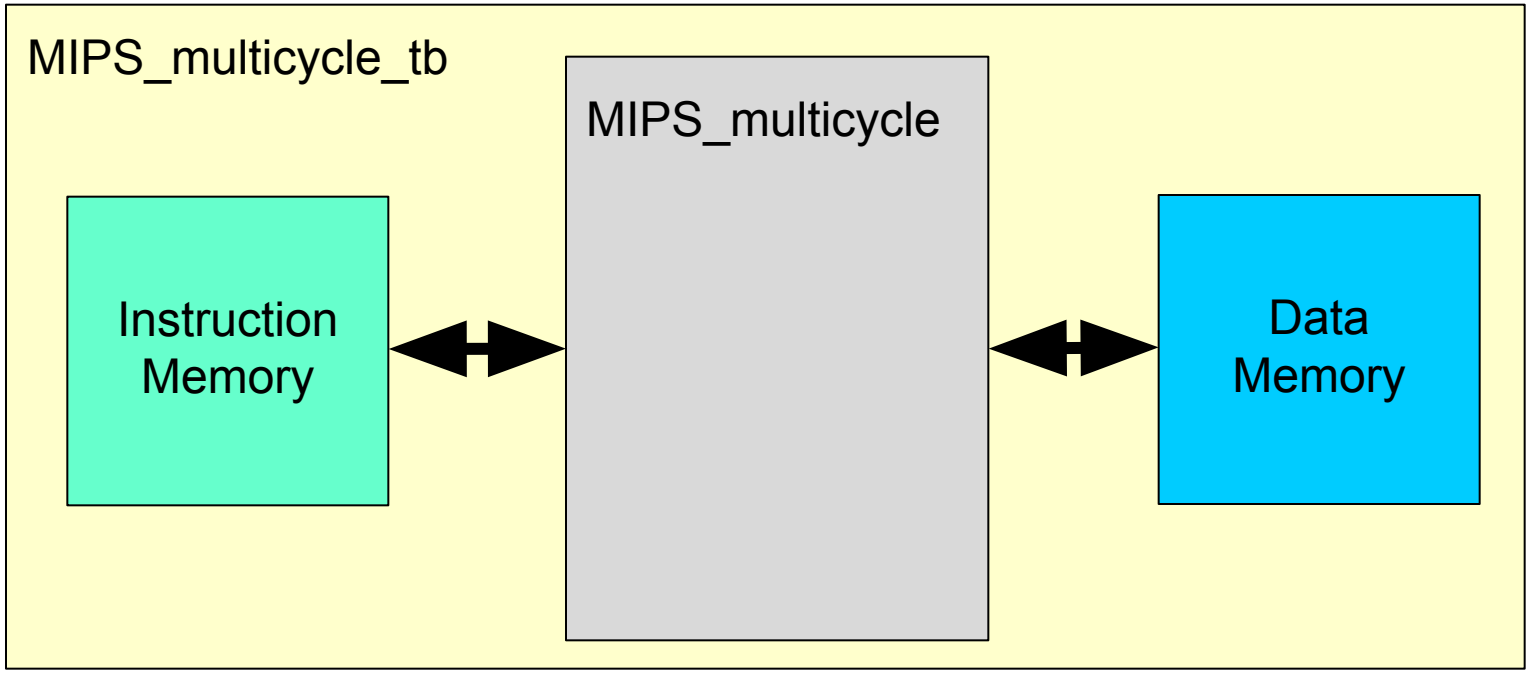
Trabalho 1 - parte 1

- Simulação



Trabalho 1 - parte 1

- MIPS Multiciclo
  - Implementar uma descrição totalmente comportamental utilizando um único *process* sensível ao *clock* e ao *reset*
    - Descrições de partes combinacionais/conexões podem estar fora do *process* (e.g. interface com a memória)
    - A implementação deve ter duas memórias como a versão monociclo



Trabalho 1 - parte 1

- Adicionar as seguintes instruções à descrição
  - BNE, XOR, XORI, NOR, ANDI, SLL, SRL, SLTU, SLTI, SLTIU, BGEZ, BLEZ, BLTZ, BGTZ, JALR
  - http://math-atlas.sourceforge.net/devel/assembly/mips-iv.pdf
- Programa de teste: inst_test.asm (/src/asm)
  - Conferir o valor dos registradores e da memória da simulação VHDL com a execução o programa no MARS
- Organização e projeto de computadores (David Patterson & John Hennessy)
- Tomar como referência o fluxo de execução das instruções apresentadas no livro onde o tempo de execução fica entre 2 e 5 ciclos de *clock*

Trabalho 1 - parte 1

- Dicas
  - Fundamental: entender a descrição comportamental fornecida
  - Implementar a busca da instrução (*fetch*)
  - Para cada instrução a ser adicionada, primeiro entender o seu funcionamento baseado em transferências entre registradores
  - Alterar o código VHDL
  - Verificar o funcionamento a partir da simulação VHDL
- Não modificar a estrutura da descrição fornecida
  - Alterar apenas a *architecture behavioral* do MIPS_monocycle