O trabalho II compreenderá um arquitetura RISC implementada em FPGA. A CPU em questão será baseada nas CPUs MIPS. No entanto, não utilizará o mesmo instruction set. Este será modificado de acordo, para cada grupo.

Características Principais:
- A Word da arquitetura é definida em 32 bits;
- Todo o sistema é implementado em pipeline;
- Todas as instruções são formadas por 4 bytes;
- Tanto a memória de programa quanto a memória de dados são de 1kWord;
- A cada Reset, o Program Counter sempre aponta para o endereço 0 da memória de programa;
- Para essa versão, o instruction set não contemplará instruções de Branch/Jump;
- Para essa versão, o módulo register file conterá apenas 16 registros ($s0 a $s7 e $t0 a $t7);
- Como não será implementado instruções de JMP ou Branch, ignore o bloco new pc;
- Como as memórias trabalharão com Words de 32 bits, o incremento do PC não é de 4 em 4 e sim, de 1 em 1.

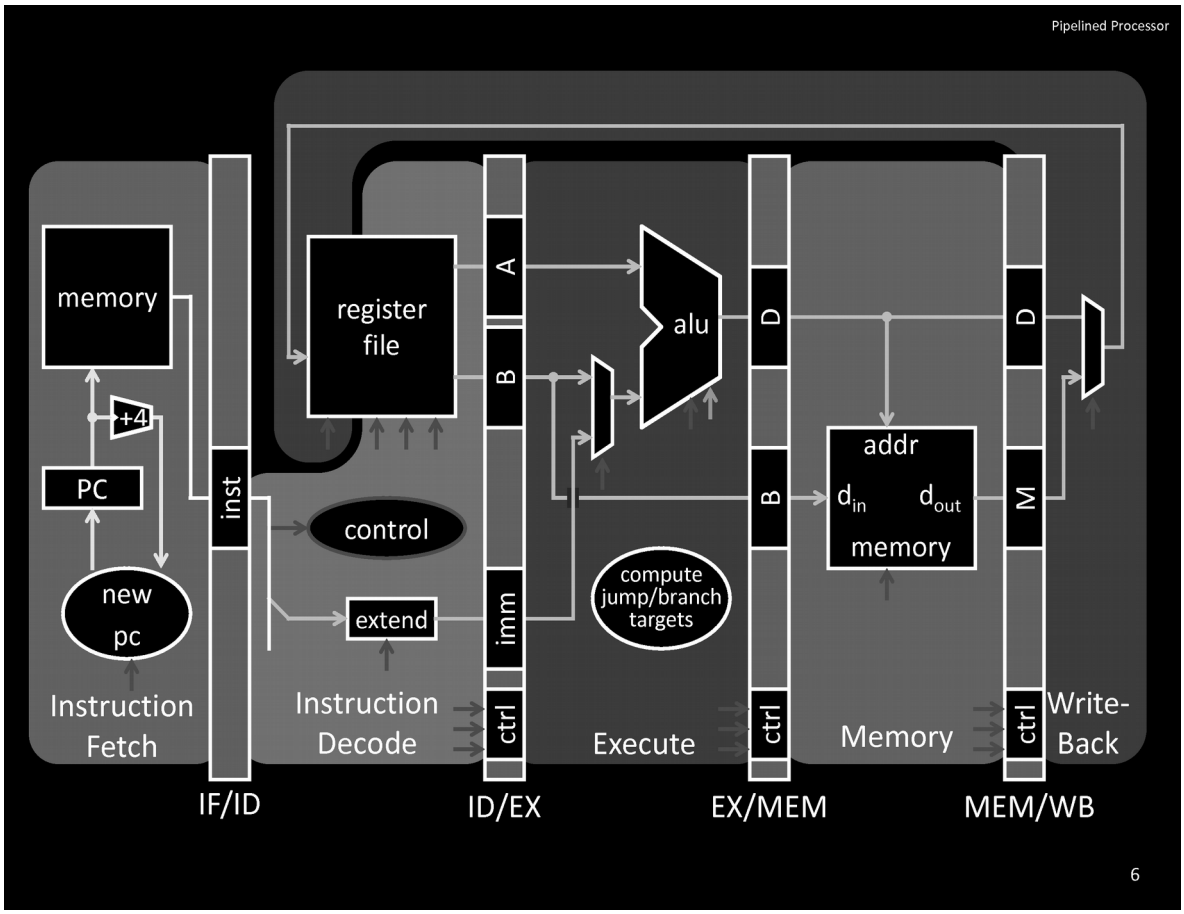

Fig. 1 – Arquitetura MIPS

A figura acima mostra a arquitetura proposta, com todos os seus componentes, e seus 5 estágios de pipeline (Instruction Fetch, Instruction Decode, Execute, Memory e Write Back).

O projeto deverá ser feito utilizando hierarquia, ou seja, cada módulo deve ser feito e testado separadamente. Uma descrição estrutural deverá conectar todos os módulos, formando a MIPS_CPU (hierarquia top), que também deverá ser testada.

Instruction Set compreenderá apenas as intruções marcadas em cinza. Os 6 primeiros bits de cada instrução, que definem seu tipo, serão definidas pelo número do grupo.

| | |
|---|---|
| rs | Primeiro registrador Fonte |
| rt | Segundo registrador Fonte |
| rd | Registrador destino |

Tab. 1 – Legenda

| Instruction name | Mnemonic | Format | Encoding (10) | | | |
|---|---|---|---|---|---|---|
| **Tamanho em Bits** | | | **6** | **5** | **5** | **16** |
| **Load Byte** | LB | I | 32 | rs | rt | offset |
| **Load Halfword** | LH | I | 33 | rs | rt | offset |
| **Load Word Left** | LWL | I | 34 | rs | rt | offset |
| **Load Word** | LW | I | (Grupo+1) | rs | rt | offset |
| **Load Byte Unsigned** | LBU | I | 36 | rs | rt | offset |
| **Load Halfword Unsigned** | LHU | I | 37 | rs | rt | offset |
| **Load Word Right** | LWR | I | 38 | rs | rt | offset |
| **Store Byte** | SB | I | 40 | rs | rt | offset |
| **Store Halfword** | SH | I | 41 | rs | rt | offset |
| **Store Word Left** | SWL | I | 42 | rs | rt | offset |
| **Store Word** | SW | I | (Grupo+2) | rs | rt | offset |
| **Store Word Right** | SWR | I | 46 | rs | rt | offset |

| Instruction name | Mnemonic | Format | Encoding (10) | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Tamanho em Bits** | | | **6** | **5** | **5** | **5** | **5** | **6** |
| **Add** | ADD | R | Grupo | rs | rt | rd | 10 | 32 |
| **Add Unsigned** | ADDU | R | 0 | rs | rt | rd | 10 | 33 |
| **Subtract** | SUB | R | Grupo | rs | rt | rd | 10 | 34 |
| **Subtract Unsigned** | SUBU | R | 0 | rs | rt | rd | 10 | 35 |
| **And** | AND | R | Grupo | rs | rt | rd | 10 | 36 |
| **Or** | OR | R | Grupo | rs | rt | rd | 10 | 37 |
| **Exclusive Or** | XOR | R | 0 | rs | rt | rd | 10 | 38 |
| **Nor** | NOR | R | 0 | rs | rt | rd | 10 | 39 |
| **Set on Less Than** | SLT | R | 0 | rs | rt | rd | 10 | 42 |
| **Set on Less Than Unsigned** | SLTU | R | 0 | rs | rt | rd | 10 | 43 |
| **Add Immediate** | ADDI | I | 8 | rs | rd | immediate | | |
| **Add Immediate Unsigned** | ADDIU | I | 9 | $s | $d | immediate | | |
| **Set on Less Than Immediate** | SLTI | I | 10 | $s | $d | immediate | | |
| Set on Less Than Immediate U | SLTIU | I | 11 | $s | $d | immediate | | |
| And Immediate | ANDI | I | 12 | $s | $d | immediate | | |
| Or Immediate | ORI | I | 13 | $s | $d | immediate | | |
| Exclusive Or Immediate | XORI | I | 14 | $s | $d | immediate | | |
| Load Upper Immediate | LUI | I | 15 | 10 | $d | immediate | | |

Tabela 2 – MIPS ISA Modificado

O Programa de TestBench da CPU deverá executar a seguinte expressão matemática:

**X = (A+B) – (C+D)**,

onde: **A=1000$_{(10)}$**, **B=5000$_{(10)}$**, **C=2000$_{(10)}$** e **D=3000$_{(10)}$**.

Obs.: Traduza a expressão acima para MIPS Assembly e após gere o código binário a ser salvo na memória de programa (utilize como base as Tabela 2 e 3). Lembre-se: arquiteturas MIPS não fazem operações aritméticas/lógicas diretamente em memória. Os dados precisam primeiramente ser carregados nos registros. Tabela 3 exemplifica a codificação Assembly MIPS

Assuma que a arquitetura seja BigEndian.

---

**Avaliação**:

Após a implementação e verificação do correto funcionamento do circuito, responda:

a) Qual a latência do sistema?

b) Qual o throughput do sistema?

c) Qual a máxima frequência de operação do sistema, considerando a implementação na Altera Cyclone II EP2C20F484C7?

d) Com a arquitetura implementada, a expressão X = (A+B) – (C+D) é executada corretamente? Por quê? O que pode ser feito para que a expressão seja calculada corretamente?

**O que entregar? Arquivo zip com a estrutura hierárquica do projeto como a seguir:**
(arquivos qws, qpf e qsf são gerados automaticamente ao se criar o projeto)

- → MIPS_CPU (pasta zipada)
  - → cpu.v (descrição estrutural ligando os módulos)
  - → TB.v (testbench da cpu)
  - → cpu.qws
  - → cpu.qpf
  - → cpu.qsf
  - → DataMemory
    - → datamemory.v
    - → datamemory_TB.v
    - → datamemory.qws
    - → datamemory.qpf
    - → datamemory.qsf
  - → InstructionMemory
    - → instructionmemory.v
    - → instructionmemory_TB.v
    - → instructionmemory.qws
    - → instructionmemory.qpf
    - → instructionmemory.qsf
  - → MUX
    - → mux.v
    - → mux_TB.v
    - → mux.qws
    - → mux.qpf
    - → mux.qsf
  - → PC
    - → pc.v
    - → pc_TB.v
    - → pc.qws
    - → pc.qpf
    - → pc.qsf
  - → ALU
    - → alu.v
    - → alu_TB.v
    - → alu.qws
    - → alu.qpf
    - → alu.qsf
  - → Control
    - → control.v
    - → control_TB.v
    - → control.qws
    - → control.qpf
    - → control.qsf
  - → RegisterFile
    - → registerfile.v
    - → registerfile_TB.v
    - → registerfile.qws
    - → registerfile.qpf
    - → registerfile.qsf
  - → Extend (estende o offset de 16 bits para 32 bits nas instruções lw e sw. Trabalharemos com offset não sinalizado. Bloco apenas concatenará 16 zeros aos 16 bits do offset).
    - → extend.v
    - → extend_TB.v
    - → extend.qws
    - → extend.qpf
    - → extend.qsf
  - → 32bitsRegister
    - → 32bitsRegister.v
    - → 32bitsRegister_TB.v
    - → 32bitsRegister.qws
    - → 32bitsRegister.qpf
    - → 32bitsRegister.qsf

**MIPS operands**

| Name | Example | Comments |
|---|---|---|
| 32 registers | $s0, $s1, . . . , $t0, $t1, . . . | Fast locations for data. In MIPS, data must be in registers to perform arithmetic. |
| $2^{30}$ memory words | Memory[0], Memory[4], . . . , Memory[4294967292] | Accessed only by data transfer instructions in MIPS. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers. |

**MIPS assembly language**

| Category | Instruction | Example | Meaning | Comments |
|---|---|---|---|---|
| Arithmetic | add | add $s1,$s2,$s3 | $s1 = $s2 + $s3 | three operands; data in registers |
| | subtract | sub $s1,$s2,$s3 | $s1 = $s2 – $s3 | three operands; data in registers |
| Data transfer | load word | lw $s1,100($s2) | $s1 = Memory[$s2 + 100] | Data from memory to register |
| | store word | sw $s1,100($s2) | Memory[$s2 + 100] = $s1 | Data from register to memory |

**FIGURE 3.4 MIPS architecture revealed through section 3.3.** Highlighted portions show MIPS assembly langua structures introduced in section 3.3.

**MIPS operands**

| Name | Example | Comments |
|---|---|---|
| 32 registers | $s0, $s1, . . . , $s7 $t0, $t1, . . . , $t7 | Fast locations for data. In MIPS, data must be in registers to perform arithmetic. Registers $s0–$s7 map to 16–23 and $t0–$t7 map to 8–15. |
| $2^{30}$ memory words | Memory[0], Memory[4], . . . , Memory[4294967292] | Accessed only by data transfer instructions in MIPS. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers. |

**MIPS assembly language**

| Category | Instruction | Example | Meaning | Comments |
|---|---|---|---|---|
| Arithmetic | add | add $s1,$s2,$s3 | $s1 = $s2 + $s3 | Three operands; data in registers |
| | subtract | sub $s1,$s2,$s3 | $s1 = $s2 - $s3 | Three operands; data in registers |
| Data transfer | load word | lw $s1,100($s2) | $s1 = Memory[$s2 + 100] | Data from memory to register |
| | store word | sw $s1,100($s2) | Memory[$s2 + 100] = $s1 | Data from register to memory |

**MIPS machine language**

| Name | Format | Example | | | | | | Comments |
|---|---|---|---|---|---|---|---|---|
| add | R | 0 | 18 | 19 | 17 | 0 | 32 | add $s1,$s2,$s3 |
| sub | R | 0 | 18 | 19 | 17 | 0 | 34 | sub $s1,$s2,$s3 |
| lw | I | 35 | 18 | 17 | 100 | | | lw $s1,100($s2) |
| sw | I | 43 | 18 | 17 | 100 | | | sw $s1,100($s2) |
| Field size | | 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits | All MIPS instructions 32 bits |
| R-format | R | op | rs | rt | rd | shamt | funct | Arithmetic instruction format |
| I-format | I | op | rs | rt | address | | | Data transfer format |

**FIGURE 3.6 MIPS architecture revealed through section 3.4.** Highlighted portions show MIPS machine language structures introduced in section 3.4. The two MIPS instruction formats so far are R and I. The first 16 bits are the same: both contain an *op* field, giving the base operation; an *rs* field, giving one of the sources; and the *rt* field, which specifies the other source operand, except for load word, where it specifies the destination register. R-format divides the last 16 bits into an *rd* field, specifying the destination register; *shamt* field, which is unused in Chapter 3 and hence always is 0; and the *funct* field, which specifies the specific operation of R-format instructions. I-format keeps the last 16 bits as a single *address* field.

Tabela 3 - Exemplos da codificação Assembly MIPS