Processo Seletivo CoreTech - Etapa 2

Problemas A, B e C

Liga Acadêmica de Hardware CoreTech

Problema A: Documentação e Avaliação

Descrição Geral

Bem-vindos novamente. A Etapa 2 do processo seletivo é composta por **três** problemas: A, B e C.

A entrega do **Problema A é obrigatória** para todas as equipes e consiste na documentação e avaliação do trabalho técnico realizado.

As equipes deverão **escolher apenas um** dos problemas técnicos para implementar:

- **Problema B:** Implementação de um subconjunto do processador PicoQuickProcessor em Verilog.
- **Problema C:** Implementação de um interpretador feito em C/C++ para a arquitetura PicoQuickProcessor.

O Problema A é, portanto, a documentação e análise da solução desenvolvida para o Problema B **ou** C.

Requisitos da Documentação (Problema A)

A equipe deve entregar um documento técnico (em formato PDF) descrevendo em detalhes a implementação da solução escolhida (B ou C).

Se a equipe escolheu o Problema B (Verilog)

A documentação deve focar no design de hardware e conter, no mínimo:

- Diagramas de bloco da arquitetura (Datapath e Controle).
- Descrição detalhada de cada módulo Verilog implementado.
- Explicação de como o ciclo de instrução (single-cycle) foi implementado.
- Detalhamento da lógica de decodificação e controle.
- Quaisquer desafios encontrados e decisões de design tomadas.

Se a equipe escolheu o Problema C (Interpretador)

A documentação deve focar na arquitetura do software e conter, no mínimo:

- Estruturas de dados escolhidas para simular a memória, os registradores e as flags.
- Descrição da lógica principal do interpretador (o ciclo de *fetch-decode-execute*).
- Detalhamento de como instruções complexas (como acesso à memória *little-endian* e desvios) foram implementadas.
- Explicação de como o *tracing* e as estatísticas de execução foram coletados.
- Quaisquer desafios encontrados e decisões de design tomadas.

Metodologia de Avaliação

A avaliação **não é binária** (i.e., "tudo ou nada"). Encorajamos fortemente que as equipes planejem seu trabalho em etapas e de forma modular.

Uma implementação parcial que executa corretamente um subconjunto das instruções (ex: apenas `mov`, `add` e `sub`) e é acompanhada por uma **excelente documentação (Problema A)** será mais bem avaliada do que uma tentativa de implementação completa que não funciona ou não é documentada.

Priorizem a qualidade e a funcionalidade incremental.

Entregáveis Individuais

Além do código (B ou C) e da documentação técnica (A) entregues pela equipe, cada integrante deve submeter uma resposta no Forms de avaliação, que será divulgado no fim do prazo desta etapa. Este deve ser preenchido de forma individual e terá confidencialidade total das respostas, devendo conter :

1. **Descrição de Responsabilidades:** Uma breve lista detalhando quais foram suas contribuições específicas para o projeto (ex: "Implementei os módulos X e Y", "Fui responsável pelo parsing da entrada e lógica de flags", "Escrevi as seções 1 e 2 da documentação").

2. **Autoavaliação:** Uma análise honesta sobre seu próprio desempenho, dificuldades e aprendizados durante o desafio.

3. **Avaliação dos Pares:** Uma breve avaliação sobre o desempenho e colaboração de cada um dos outros membros da sua equipe.

Problema B: Processador PicoQuick (Opção 1)

Descrição do Problema

Nesta fase, o desafio é projetar e implementar uma Unidade Central de Processamento (CPU) funcional em Verilog, baseada na arquitetura **PicoQuickProcessor**, um processador didático de 32 bits.

O objetivo é implementar um **subconjunto (subset)** do seu Conjunto de Instruções (ISA), focando nas operações fundamentais de movimentação de dados, aritmética, lógica e controle de fluxo.

Especificação da Arquitetura

A arquitetura do PicoQuickProcessor possui as seguintes características:

- **Arquitetura:** 32 bits, *little-endian*.
- **Registradores:** 16 registradores de propósito geral (GPRs) de 32 bits, nomeados de `r0` a `r15`.
- **Memória:** 256 bytes, com arquitetura Von Neumann (instruções e dados compartilham o mesmo espaço de endereçamento).
- **PC:** Um registrador interno de 32 bits (Contador de Programa) que aponta para a próxima instrução a ser buscada.

Conjunto de Instruções (Subset)

Vocês devem implementar **apenas** as instruções listadas na Tabela 1. O processador deve ser capaz de decodificar e executar estas instruções corretamente.

Tabela 1: Subconjunto de Instruções do PicoQuickProcessor

Instrução	Opcode	Campos	Semântica
`mov rx, i16`	`0x00`	$r_{x[3-0]}$, $i_{[15-0]}$	$r_x = \text{sign\_extend}(i_{15-0})$
`mov rx, ry`	`0x01`	$r_{x[3-0]}$, $r_{y[3-0]}$	$r_x = r_y$
`mov rx, [ry]`	`0x02`	$r_{x[3-0]}$, $r_{y[3-0]}$	$r_x = \text{MEM}[r_y]$ (Load)
`mov [rx], ry`	`0x03`	$r_{x[3-0]}$, $r_{y[3-0]}$	$\text{MEM}[r_x] = r_y$ (Store)
`jmp i16`	`0x05`	$i_{[15-0]}$	$pc = pc + 4 + \text{sign\_extend}(i_{15-0})$
`add rx, ry`	`0x09`	$r_{x[3-0]}$, $r_{y[3-0]}$	$r_x = r_x + r_y$
`sub rx, ry`	`0x0A`	$r_{x[3-0]}$, $r_{y[3-0]}$	$r_x = r_x - r_y$
`and rx, ry`	`0x0B`	$r_{x[3-0]}$, $r_{y[3-0]}$	$r_x = r_x \ \& \ r_y$
`or rx, ry`	`0x0C`	$r_{x[3-0]}$, $r_{y[3-0]}$	$r_x = r_x$ — r_y
`halt`	`0xFF`	–	Termina a execução

Notas sobre as Instruções (Problema B)

- r_x e r_y referem-se aos operandos de registrador (4 bits para selecionar um dos 16 GPRs).
- i_{15-0} é um valor imediato de 16 bits. Note que a instrução `mov rx, i16` deve realizar uma extensão de sinal (*sign extension*) para 32 bits.
- O desvio (`jmp`) é relativo ao PC (Program Counter).

Execução e Término (Problema B)

O processador deve ser implementado como um **processador de ciclo único** (*single-cycle*).

Neste design, cada instrução completa todas as suas etapas (busca, decodificação, execução, acesso à memória e escrita de volta) em **um único ciclo de clock**. O Contador de Programa (PC) é atualizado ao final de cada ciclo para apontar para a próxima instrução (normalmente $PC + 4$, ou o endereço de desvio no caso de um `jmp`).

A execução do processador é controlada por sinais de *clock* e *reset*. O *reset* deve inicializar o PC (para o endereço `0x00`) e os registradores.

A execução do programa deve ser interrompida (Halt) permanentemente quando a instrução `halt` (opcode `0xFF`) for buscada e decodificada. O processador deve parar de buscar novas instruções.

Entrada (Testbench - Problema B)

Não há uma entrada de dados padrão (como `stdin`). A "entrada" para este problema é o seu próprio *testbench* Verilog. O testbench deve inicializar a memória do processador com um pequeno programa de teste que valide o comportamento de **todas** as instruções do subset. O programa deve executar e, por fim, invocar a instrução `halt`.

Saída (Entregáveis - Problema B)

A saída esperada consiste em dois arquivos:

- O código-fonte completo em Verilog (`.v`) do PicoQuickProcessor.
- O código-fonte do *testbench* (`.v`) utilizado para validar o processador.

Problema C: Interpretador PicoQuick (Opção 2)

Descrição do Problema

Nesta fase, o desafio muda de hardware para software. Vocês devem implementar um **interpretador** funcional em C ou C++ para a arquitetura **PicoQuickProcessor (PQP)**.

O objetivo é criar um programa, executando em uma arquitetura x86_64, que simule o comportamento completo do PQP, lendo seu código de máquina de um arquivo, executando-o e reportando um trace detalhado da execução e o estado final do processador.

Especificação da Arquitetura

A arquitetura do PicoQuickProcessor possui as seguintes características:

- **Arquitetura:** 32 bits, *little-endian*.
- **Registradores:** 16 registradores de propósito geral (GPRs) de 32 bits, nomeados de `r0` a `r15`.
- **Memória:** 256 bytes (endereços `0x00` a `0xFF`), com arquitetura Von Neumann.
- **PC:** Um registrador interno de 32 bits (Contador de Programa) que aponta para a próxima instrução.
- **Flags:** 3 flags de condição de 1 bit: `g` (maior), `l` (menor), `e` (igual). Elas são atualizadas **apenas** pela instrução `cmp`.

Conjunto de Instruções (Completo)

Vocês devem implementar **todas** as instruções listadas na Tabela 2.

Notas sobre as Instruções (Problema C)

- **Little-Endian:** O PQP armazena dados em formato little-endian. Uma `store` (ex: `mov [r0], r1`) de `0x12345678` no endereço `0x10` escreverá `0x78` em `0x10`, `0x56` em `0x11`, `0x34` em `0x12` e `0x12` em `0x13`. O `load` (`mov rx, [ry]`) faz o inverso.
- **Imediatos:** i_{15-0} deve sofrer extensão de sinal para 32 bits em `mov` e em todos os desvios. i_{4-0} é um valor de 5 bits sem sinal.
- **Desvios (Jumps):** Todos os desvios são relativos ao PC (Program Counter) da *próxima* instrução ($PC + 4$).
- **Flags:** Apenas `cmp` afeta as flags. `jg`, `jl`, `je` apenas as leem.

Execução e Término (Problema C)

O interpretador deve simular o ciclo de execução do PQP. O PC inicia em `0x00`. Em um loop, o interpretador deve:

1. Buscar os 4 bytes da instrução na memória, no endereço apontado pelo PC.
2. Decodificar a instrução (opcode e operandos).
3. Executar a operação (lendo/escrevendo em registradores, memória ou flags).
4. Atualizar o PC para a próxima instrução (normalmente $PC + 4$, ou o alvo do desvio, se tomado).

A simulação é interrompida **imediatamente** se o PC for atualizado para o endereço especial `0xF0F0`. Nenhum trace de registradores ou contagem é impresso para este endereço; apenas a mensagem de `EXIT` é registrada.

Entrada (Problema C)

A entrada é um único arquivo de texto, `entrada.txt`. O arquivo contém o programa PQP, representado por bytes em hexadecimal, separados por espaços ou quebras de linha. O seu interpretador deve ler este arquivo e carregar os bytes na memória simulada de 256 bytes, sequencialmente, a partir do endereço `0x00`.

Saída (Problema C)

A saída deve ser um único arquivo de texto, `saida.txt`. A saída consiste em quatro partes, geradas nesta ordem:

1. **Trace de Execução:** Para cada instrução, na **primeira vez** que ela é executada em um determinado endereço (PC), uma linha de trace detalhado deve ser impressa. Se a mesma instrução (no mesmo endereço) for executada novamente, nada deve ser impresso.

Tabela 2: Conjunto de Instruções PicoQuickProcessor (Completo)

Instrução	Opcode	Campos	Semântica
`mov rx, i16`	0x00	$r_{x[3-0]}$, $i_{[15-0]}$	$r_x = \text{sign\_extend}(i_{15-0})$
`mov rx, ry`	0x01	$r_{x[3-0]}$, $r_{y[3-0]}$	$r_x = r_y$
`mov rx, [ry]`	0x02	$r_{x[3-0]}$, $r_{y[3-0]}$	$r_x = \text{MEM}[r_y]$ (Load 32-bit, little-endian)
`mov [rx], ry`	0x03	$r_{x[3-0]}$, $r_{y[3-0]}$	$\text{MEM}[r_x] = r_y$ (Store 32-bit, little-endian)
`cmp rx, ry`	0x04	$r_{x[3-0]}$, $r_{y[3-0]}$	Compara r_x, r_y (com sinal) e atualiza flags `g`, `l`, `e`
`jmp i16`	0x05	$i_{[15-0]}$	$pc = pc + 4 + \text{sign\_extend}(i_{15-0})$
`jg i16`	0x06	$i_{[15-0]}$	Se `g=1`, $pc = pc + 4 + \text{sign\_extend}(i_{15-0})$
`jl i16`	0x07	$i_{[15-0]}$	Se `l=1`, $pc = pc + 4 + \text{sign\_extend}(i_{15-0})$
`je i16`	0x08	$i_{[15-0]}$	Se `e=1`, $pc = pc + 4 + \text{sign\_extend}(i_{15-0})$
`add rx, ry`	0x09	$r_{x[3-0]}$, $r_{y[3-0]}$	$r_x = r_x + r_y$
`sub rx, ry`	0x0A	$r_{x[3-0]}$, $r_{y[3-0]}$	$r_x = r_x - r_y$
`and rx, ry`	0x0B	$r_{x[3-0]}$, $r_{y[3-0]}$	$r_x = r_x \ \& \ r_y$
`or rx, ry`	0x0C	$r_{x[3-0]}$, $r_{y[3-0]}$	$r_x = r_x$ — r_y
`xor rx, ry`	0x0D	$r_{x[3-0]}$, $r_{y[3-0]}$	$r_x = r_x \ \hat{} r_y$
`sal rx, i5`	0x0E	$r_{x[3-0]}$, $i_{[4-0]}$	$r_x = r_x \ll i_5$ (Shift Lógico)
`sar rx, i5`	0x0F	$r_{x[3-0]}$, $i_{[4-0]}$	$r_x = r_x \gg i_5$ (Shift Aritmético)

2. **Término:** Ao pular para o endereço `0xF0F0`, a linha `0xF0F0->EXIT` deve ser impressa.

3. **Estatísticas de Opcodes:** Imediatamente após a linha de EXIT, imprima um sumário (em uma única linha) da contagem total de execução para *cada opcode* (0x00 a 0x0F), no formato `[opcode:contagem,...]`.

4. **Panorama dos Registradores:** Na última linha, imprima os valores finais (em hexadecimal) de todos os 16 registradores, no formato `[R0=valor,...]`.

Exemplo de Entrada (entrada.txt)

```
0x05 0x00 0x04 0x00
0xCA 0xF0 0xF0 0x50
0x00 0x00 0x04 0x00
0x02 0x10 0x00 0x00
0x00 0x00 0xFF 0x7F
0x0E 0x00 0x00 0x01
0x00 0xF0 0x01 0x00
0x09 0x0F 0x00 0x00
0x0E 0x00 0x00 0x08
0x0B 0x10 0x00 0x00
0x0F 0x10 0x00 0x08
0x00 0xF0 0x04 0x00
0x03 0xF1 0x00 0x00
0x04 0x8F 0x00 0x00
0x07 0x00 0x00 0x00
0x0A 0xF1 0x00 0x00
0x04 0x8F 0x00 0x00
0x06 0x00 0x00 0x00
0x0C 0x8F 0x00 0x00
0x00 0x80 0xE4 0xFF
0x0D 0x8F 0x00 0x00
0x04 0xDD 0x00 0x00
0x08 0x00 0x04 0x00
0xEB 0x2B 0x62 0x01
0x00 0x10 0x5C 0x00
0x02 0xF1 0x00 0x00
0x0D 0x11 0x00 0x00
0x0D 0x88 0x00 0x00
0x00 0xB0 0x00 0x00
0x00 0xC0 0x01 0x00
0x01 0xDB 0x00 0x00
0x09 0xDC 0x00 0x00
0x01 0x01 0x00 0x00
0x01 0x12 0x00 0x00
0x01 0x23 0x00 0x00
0x01 0x34 0x00 0x00
0x01 0x45 0x00 0x00
0x01 0x56 0x00 0x00
0x01 0x67 0x00 0x00
0x01 0x78 0x00 0x00
0x01 0x89 0x00 0x00
0x01 0x9A 0x00 0x00
0x01 0xAB 0x00 0x00
0x01 0xBC 0x00 0x00
0x01 0xCD 0x00 0x00
0x00 0xE0 0x01 0x00
0x0E 0x50 0x00 0x00
0x0F 0x70 0x00 0x00
0x0B 0x88 0x00 0x00
0x0C 0xFF 0x00 0x00
0x0A 0xFE 0x00 0x00
0x04 0xFE 0x00 0x00
0x08 0x00 0x00 0x00
0x07 0x00 0x00 0x00
0x05 0x00 0x00 0x00
0x06 0x00 0x98 0xFF
0x00 0xF0 0x5C 0x00
0x02 0xDF 0x00 0x00
0x04 0xDE 0x00 0x00
0x08 0x00 0x00 0xF0
0x0A 0xDE 0x00 0x00
0x03 0xFD 0x00 0x00
0x05 0x00 0xE8 0xFF
```

```
0x78 0x56 0x34 0x12
\end{T>
```

Exemplo de Saída (saida.txt)

```
0x0000->JMP_0x0008
0x0008->MOV_R0=0x00000004
0x000C->MOV_R1=MEM[0x04,0x05,0x06,0x07]=[0xCA,0xF0,0xF0,0x50]
0x0010->MOV_R0=0x00007FFF
0x0014->SAL_R0<<=1=0x00007FFF<<1=0x0000FFFE
0x0018->MOV_R15=0x00000001
0x001C->ADD_R0+=R15=0x0000FFFE+0x00000001=0x0000FFFF
0x0020->SAL_R0<<=8=0x0000FFFF<<8=0x00FFFF00
0x0024->AND_R1&=R0=0x50F0F0CA&0x00FFFF00=0x00F0F000
0x0028->SAR_R1>>=8=0x00F0F000>>8=0x0000F0F0
0x002C->MOV_R15=0x00000004
0x0030->MOV_MEM[0x04,0x05,0x06,0x07]=R1=[0xF0,0xF0,0x00,0x00]
0x0034->CMP_R8<=>R15(G=0,L=1,E=0)
0x0038->JL_0x003C
0x003C->SUB_R15-=R1=0x00000004-0x0000F0F0=0xFFFF0F14
0x0040->CMP_R8<=>R15(G=1,L=0,E=0)
0x0044->JG_0x0048
0x0048->OR_R8|=R15=0x00000000|0xFFFF0F14=0xFFFF0F14
0x004C->MOV_R8=0xFFFFFFE4
0x0050->XOR_R8^=R15=0xFFFFFFE4^0xFFFF0F14=0x0000F0F0
0x0054->CMP_R13<=>R13(G=0,L=0,E=1)
0x0058->JE_0x0060
0x0060->MOV_R1=0x0000005C
0x0064->MOV_R15=MEM[0x5C,0x5D,0x5E,0x5F]=[0xEB,0x2B,0x62,0x01]
0x0068->XOR_R1^=R1=0x0000005C^0x0000005C=0x00000000
0x006C->XOR_R8^=R8=0x0000F0F0^0x0000F0F0=0x00000000
... (saída do trace continua) ...
0x00F8->JMP_0x00E4
0xF0F0->EXIT
[00:23210995,01:324953804,02:23210989,03:23210987,04:46421976,05:46421973,06:23210987,07:23210987,08:46421974,09:23210987,0A:464
[R0=0x177B38C1,R1=0x4B645E80,R2=0x62DF9741,R3=0xAE43F5C1,R4=0x11238D02,R5=0xBF6782C3,R6=0xD08B0FC5,R7=0x8FF29288,R8=0x607DA24D,R
```